ENCICLOPEDIA PRACTICA DE LA

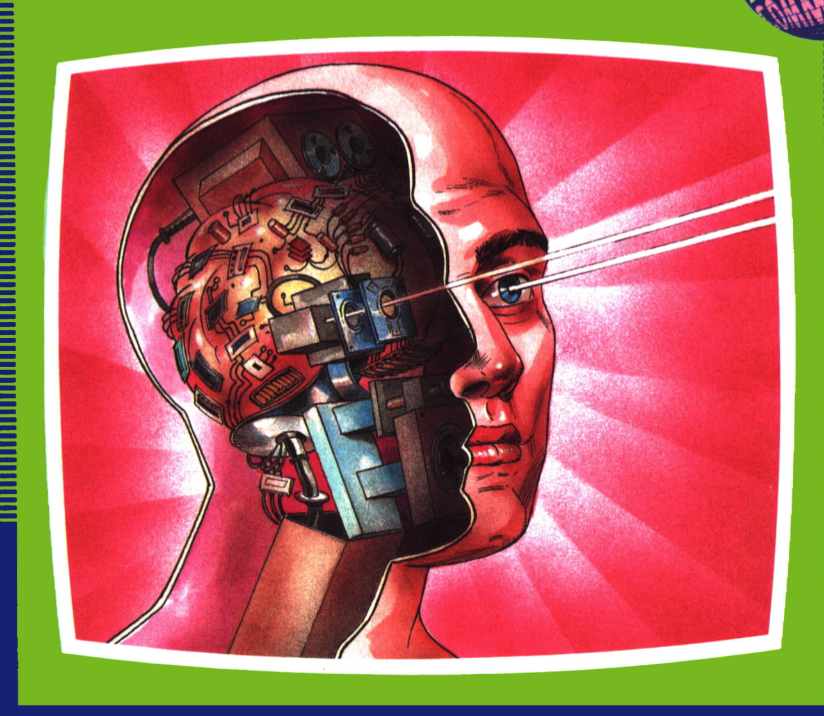
INFORMATICA APLICADA

17

Sistemas operativos: el sistema nervioso del ordenador

Aula de Informática

SINCLAIR AMSTRAD COMMODORE IBM



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

17

Sistemas operativos: el sistema nervioso del ordenador

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación
y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo XVII. **Sistemas operativos: el sistema nervioso del ordenador.**
AULA DE INFORMATICA APLICADA
ENRIQUE SERRANO, Ingeniero de Telecomunicación
VICENTE NEIRA, Técnico de Informática

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Sor Angela de la Cruz, 24-7.º G. Teléf. 279 40 36. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. San Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-060-X
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 2.543-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Sor Angela de la Cruz, 24-7.° G. Teléf. 279 40 36. 28020 Madrid

Octubre, 1986.
P.V.P. Canarias: 365,-

INDICE

1	Introducción	5
2	Cargador y montador («loader» y «Linkage editor»)	11
3	Interrupciones	21
4	Operaciones de entrada/salida	29
5	Multiprogramación	39
6	Funciones adicionales de los sistemas operativos	49
7	Introducción al DOS	57
8	Comandos del DOS	59
9	Comandos de proceso por lotes	71
10	Utilidades	75
11	Configuración	79
12	Introducción al CP/M	83
13	Comandos del CP/M	85

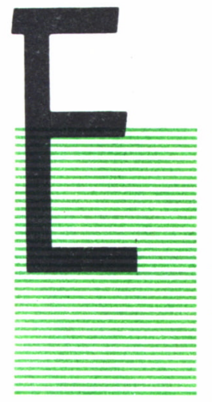
Los programas que aparecen en este libro funcionan en los ordenadores:

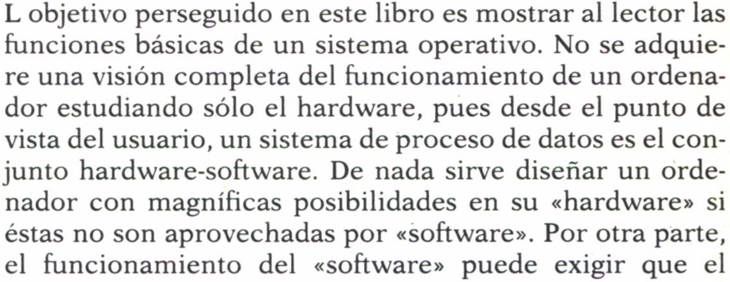
IBM–PC, XT, AT y compatibles.
AMSTRAD–464, 664, 6128, 1512.
SINCLAIR–SPECTRUM 48 K, 128 K, PLUS, PLUS 2.
MSX–Todos los modelos.
COMMODORE–CBM 64 y CBM 128.

INTRODUCCION 1



OBJETIVO





EL objetivo perseguido en este libro es mostrar al lector las funciones básicas de un sistema operativo. No se adquiere una visión completa del funcionamiento de un ordenador estudiando sólo el hardware, pues desde el punto de vista del usuario, un sistema de proceso de datos es el conjunto hardware-software. De nada sirve diseñar un ordenador con magníficas posibilidades en su «hardware» si éstas no son aprovechadas por «software». Por otra parte, el funcionamiento del «software» puede exigir que el «hardware» esté dotado de ciertos dispositivos.

En realidad, en el diseño de un ordenador deben intervenir especialistas de «software» junto a los de «hardware».

En la primera parte de este libro pretendemos dar unas nociones básicas sobre lo que es un sistema operativo, reservando la parte final para una breve descripción de los sistemas operativos más utilizados en la gama de los ordenadores personales. Nos estamos refiriendo al CP/M, de Digital Research, y al MS-DOS, de Microsoft.

Para el estudio de un sistema operativo introduciremos las ideas fundamentales valiéndonos de un modelo de ordenador sencillo, y basándonos en él iremos desarrollando los distintos aspectos de un sistema operativo.

Se han respetado algunos términos ingleses para facilitar la comprensión de la abundante literatura que hay en inglés sobre estos temas.



ALGUNOS CONCEPTOS BASICOS Y DEFINICIONES

Repasemos algunos conceptos.

Hardware es el conjunto de máquinas y circuitería que forman un ordenador (UCP, canales, memoria, dispositivos de E/S, etc.).

Al conjunto de todos los programas disponibles para un ordenador le llamaremos *software*.

Dentro del software distinguiremos dos clases de programas:

1. *Software general*

Conjunto de programas de uso generalizado, de uso disponible para toda la comunidad de usuarios de un tipo de ordenador. Cuando hablamos de software, sin más, nos referimos, naturalmente, a esta clase de programas. En él se incluyen el sistema operativo y los programas generalizados de aplicación (por ejemplo: programas disponibles en el mercado para cálculos estadísticos, control de producción, programación lineal, cálculo de estructuras, etc.).

2. *Software privado*

Programas desarrollados por alguna persona o entidad privada para su propio uso y no disponible en el mercado.

Recordemos los componentes básicos de un ordenador:

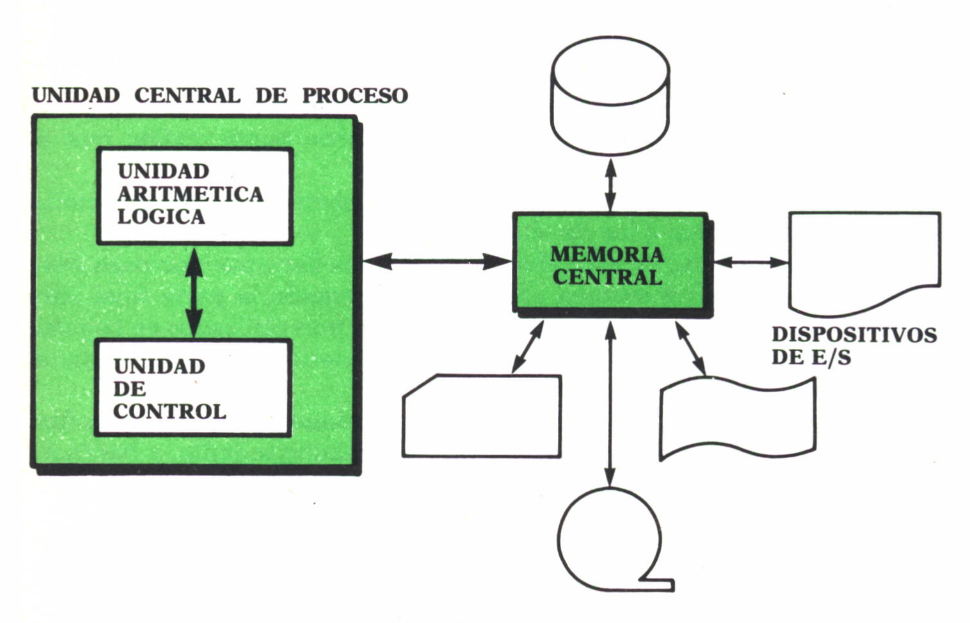


Fig. 1. Estructura básica de un ordenador y sus periféricos.

Las instrucciones de un programa y sus datos se almacenan en la memoria central. Una vez allí, el programa es ejecutado por la unidad central.

Las instrucciones son ejecutadas por la unidad de control, una a una, y en la secuencia en que están almacenadas en memoria.



¿QUE ES UN SISTEMA OPERATIVO?

Un sistema operativo es un conjunto de programas que ayudan al usuario a obtener una operación y programación más eficiente. Normalmente el sistema operativo se tiene almacenado en un disco y sus componentes más importantes son:

— Programa supervisor.
— Programa cargador («loader») (forma parte del *supervisor*).
— Programa montador («linkage editor»).
— Programas traductores de lenguaje (ensamblador normalmente; a veces también se incluyen otros lenguajes.
— Programas de «utilidad» para ficheros («File Utility Management Programs»).



OBJETIVOS DE UN SISTEMA OPERATIVO

Un sistema operativo es un conjunto de programas, normalmente proporcionado al usuario por el constructor del ordenador.

El objetivo perseguido por estos programas es facilitar la utilización del ordenador a todo el equipo humano que lo usa, es decir, a programadores y operadores fundamentalmente.

Lo que se pretende, en definitiva, es mejorar la utilización del ordenador optimizando el uso de todos sus recursos: unidad central, memoria central y dispositivos de entrada/salida (E/S).

De esta forma se obtendrá el máximo rendimiento y rentabilidad a la inversión realizada en el ordenador.

Para ello, los programas que componen el sistema operativo deberán llevar un control lo más exhaustivo posible del estado instantáneo de cada recurso, distribuyéndolos y asignándolos en cada momento, de modo que su utilización sea lo mejor posible. A lo largo de este libro iremos describiendo, de forma básica, cómo se consigue esto.



DESCRIPCION GENERAL DE UN SISTEMA OPERATIVO

Como ya hemos dicho, un sistema operativo es un conjunto de programas y rutinas. Están almacenados en un disco, normalmente, y pueden agruparse en dos categorías:

1. Los programas que forman el programa de control, también llamado *supervisor*.

2. Los programas auxiliares.

Al arrancar el trabajo del día del ordenador lo primero que hace el operador es montar el disco donde residen las rutinas del sistema operativo, y cargar en memoria, a partir de él, el programa de control o *supervisor*.

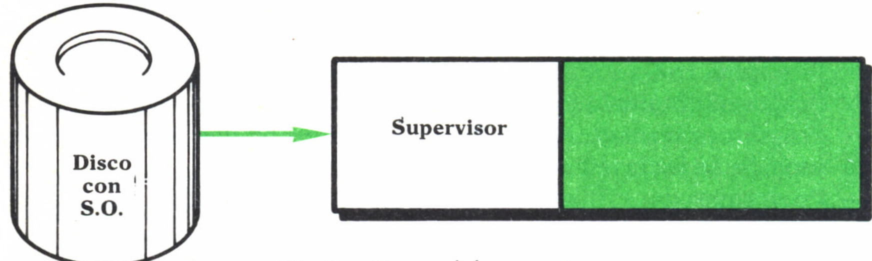


Fig. 2. Carga del supervisor.

Esta operación suele denominarse «hacer IPL» (IPL = Initial Program Loading). Normalmente basta apretar una tecla del panel del ordenador para hacer IPL (tecla «LOAD»).

Una vez que el programa *supervisor* o de control esté cargado, empieza a ejecutarse, pidiendo al operador datos sobre el primer trabajo a realizar, mediante mensajes que aparecen en la consola. El operador se los proporciona (por ejemplo: nombre del programa a ejecutar; si necesita fichas en disco o cinta; en qué unidades van a estar éstos; qué nombres tiene, etc.). Normalmente, el programa a ejecutar se encuentra, ya compilado y en código máquina ejecutable, en un fichero en disco (estos ficheros que contienen programas suelen llamarse librerías).

Una vez que el *supervisor* ha recibido del operador la identificación del programa a ejecutar (nombre y librería en que se encuentra) y de los recursos que necesita (discos y cintas), el *supervisor* lee de la librería citada el código máquina del programa y lo carga en memoria.

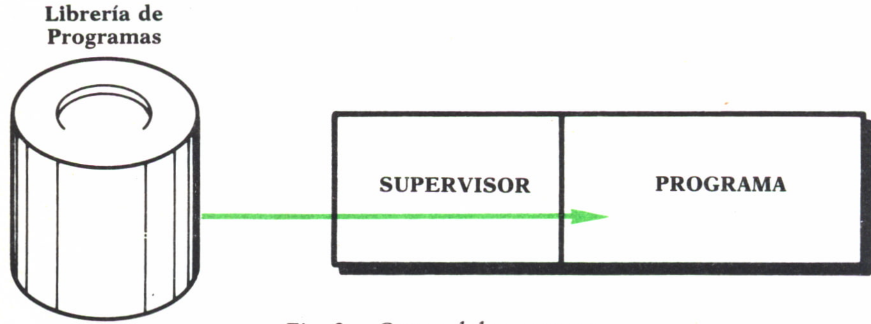


Fig. 3. Carga del programa.

Entonces, mira si las cintas y discos necesarios están ya montados en sus unidades respectivas. Sí así no fuera, dará mensajes al operador pidiéndoselos. Este los montará, preparará otros dispositivos si son necesarios (por ejemplo: fichas de datos, fichas en blanco para ser perforadas, cintas de papel, etc.), y, mediante un mensaje al *supervisor,* notificará a éste que ya está todo listo. El *supervisor* entonces transfiere control a la primera instrucción del programa a ejecutar. A partir de aquí se ejecuta éste, una instrucción tras otra, hasta que termina. Entonces no da una instrucción de parar, sino que devuelve control al *supervisor.* Este comunica al operador que ha terminado la ejecución del programa y le da orden de desmontar las cintas y discos utilizados.

Ahora, el operador puede volver a solicitar la ejecución de otro programa, y así sucesivamente.

De la descripción anterior destacamos los siguientes hechos:

1. El *supervisor* se carga en memoria al principio del día y permanece en ella, activo, durante todos los trabajos.
2. El *supervisor* se encarga de comunicarse con el operador (normalmente a través de la consola del operador).
3. El *supervisor* comprueba que es correcto el montaje de los ficheros que el programa necesita.

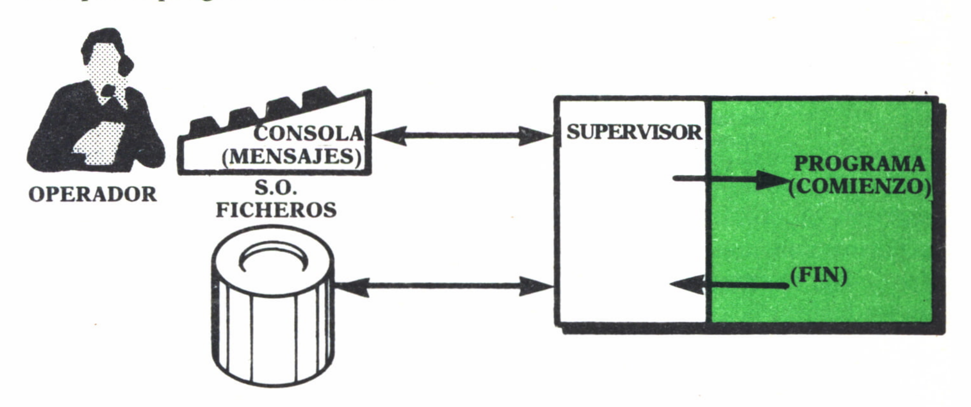


Fig. 4. Actividades del supervisor.

Hasta aquí el *supervisor* ayuda sólo al operador, haciéndole más fluido y seguro su trabajo. Pero ¿no podría ayudar al programador también? Sí, ahorrándole trabajo. Para ello, algunas rutinas que se utilizan mucho, prácticamente en todos los programas, se incluyen en el programa *supervisor,* con lo cual el programador no tiene que hacerlas, sino que basta que transfiera control al *supervisor* cuando las necesite.

Cuando el programador quiere leer un registro basta que se lo pida al *supervisor,* indicándole de qué dispositivo quiere leer y en qué área de me-

moria de su programa quiere que le proporcionen los datos. El *supervisor* da las instrucciones máquina necesarias, según el dispositivo de que se trate, recuperando errores si los hubiere y fuera posible (por ejemplo, reintenta la lectura de una cinta si detecta error de paridad), y una vez leídos los datos, los mueve al área de memoria que le han indicado, devolviendo el control al programa.

Además de las rutinas de entrada/salida (E/S), hay otras en el *supervisor* que pueden ser invocadas por el programa, de modo que durante la ejecución de éste hay una frecuente transferencia de datos y de control entre él y el *supervisor*.

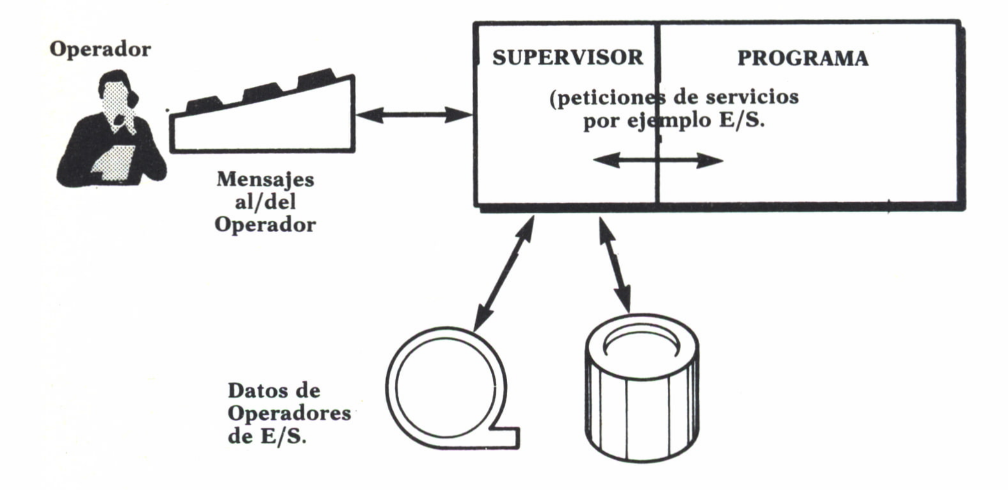
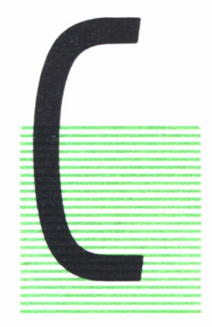


Fig. 5. Transferencia de datos entre el programa y el supervisor.

Otra función del *supervisor* es detectar situaciones anómalas o erróneas, y comunicarlas al operador para que éste tome las medidas oportunas. Ejemplos de situaciones anómalas pueden ser: al cargar el programa se detecta que necesita más memoria de la que hay disponible, una cinta da error irrecuperable de lectura, una unidad de discos está apagada, etc.

CARGADOR Y MONTADOR («LOADER» Y «LINKAGE EDITOR») 2



OMO se dijo en el capítulo anterior, una de las subrutinas que forman parte del programa *supervisor,* y que, por tanto, residirán permanentemente en memoria, es la rutina «loader». Su misión es cargar en memoria los programas e iniciar su ejecución, de acuerdo con las órdenes recibidas del operador a través del teclado de la consola.

REPASO DE ALGUNOS CONCEPTOS DE PROGRAMACION

Para resolver un determinado problema con un ordenador, el primer paso es analizarlo estudiando el método a emplear en su resolución y qué información de entrada y de salida se necesitará. El resultado de esta fase de análisis del problema suele plasmarse en su organigrama y en unos formularios para toma de datos y para impresión de resultados.

La siguiente fase es escribir el programa. Este trabajo se realiza en un determinado lenguaje de programación, por ejemplo: Fortran, Algol, PL/I, Cobol o Ensamblador. El resultado de esta fase de programación es una serie de sentencias codificadas en el lenguaje elegido. Estas sentencias se trasladan del papel a otro soporte legible por el ordenador, produciendo como resultado el «programa fuente». Los soportes empleados más frecuentemente son discos, cintas, fichas y también a veces cinta de papel.

La siguiente fase corre a cargo del ordenador. Este, mediante la ejecución de un programa traductor o compilador, lee el «programa fuente» y lo traduce a códigos de máquina ejecutable. El programa obtenido, ya en código de máquina, se llama «programa objeto», y suele producirse en un soporte legible por el ordenador que normalmente suele ser disco, y a veces ficha perforada o cinta de papel, además de listarlo para que el programador pueda trabajar con él.



CARGADOR (LOADER)

Supongamos que ya hemos compilado un programa y lo tenemos en código máquina ejecutable. Supongamos que este «programa objeto» lo tenemos en fichas perforadas (sería igual si estuviera grabado en disco magnético o perforado en cinta de papel). Tenemos el siguiente problema: ¿cómo introducimos el programa objeto en memoria para ejecutarlo? La solución es disponer de un programa o subrutina llamada «loader», cuya función es leer las fichas que componen el programa objeto, en posiciones contiguas de memoria, y luego transferir control a la primera posición de memoria ocupada. Un organigrama general de «loader» podría ser:

Comienzo
Almacenar en L la dirección de memoria donde se quiere cargar el programa.
Leer ficha del programa objeto.
Mover datos de la ficha leída a la posición de memoria cuya dirección está en L.
Sumar 80 al contenido de L.
Era la última ficha del programa objeto ?
NO
SI
Transferir control a la primera posición del programa cargado en memoria.
FIN

Fig. 6. Organigrama general de «loader».

El organigrama anterior habría que completarlo con alguna comprobación para evitar algunos posibles errores. Por ejemplo: las fichas pueden llevar perforada en ciertas columnas una identificación del programa objeto (un nombre cualquiera elegido por el programador) y en otras columnas un número de secuencia (que tendrá perforado 1 en la primera ficha, 2 en la segunda ficha, etc.). De este modo el «loader» podría comprobar que todas las fichas leídas pertenecen al mismo programa, que todas están en el orden adecuado y que no falta ninguna.

El organigrama general podría ser así (suponiendo que la identificación ocupa seis columnas de la ficha y la secuencia cuatro, y que quedan, por tanto, 70 columnas de información).

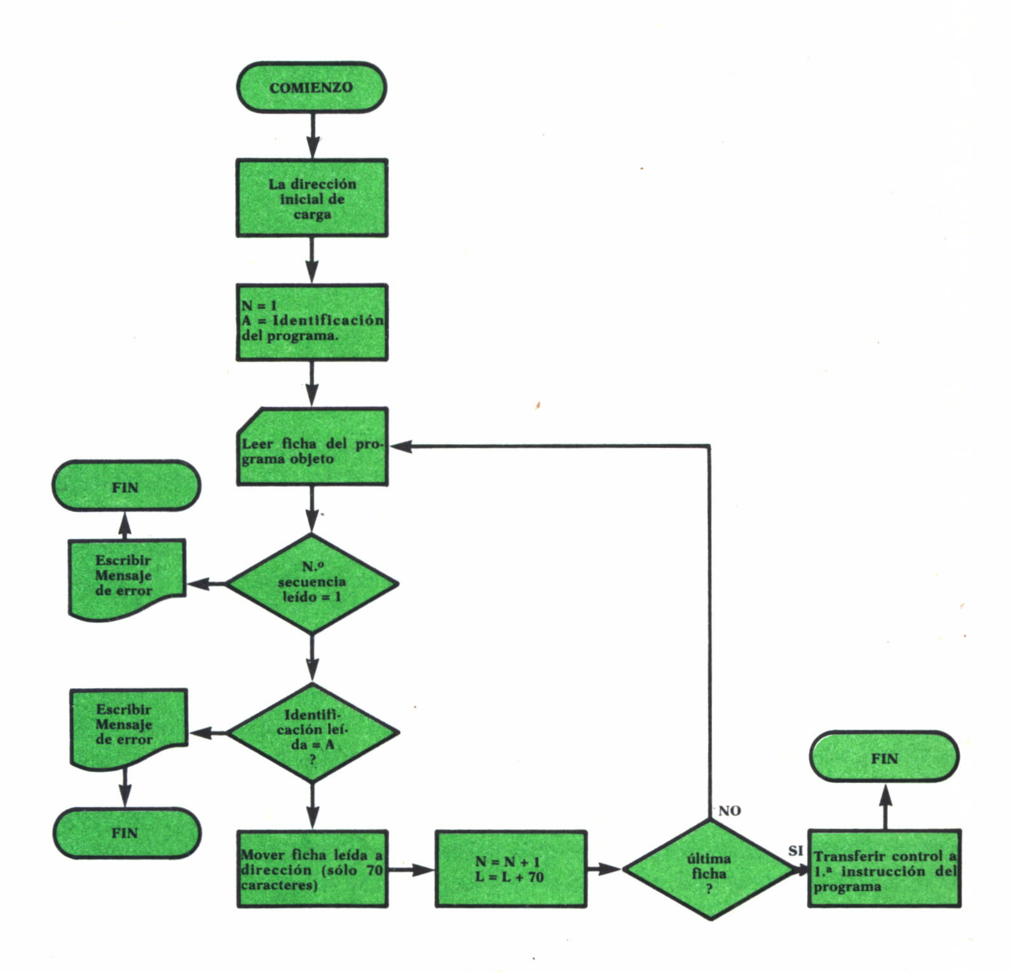


Fig. 7. Organigrama general.

REUBICACION DE PROGRAMAS (PROGRAM RELOCATION)

Como se dijo anteriormente, el *supervisor* está permanentemente en memoria. Si éste ocupa 4.000 palabras (es decir, de la 0 a la 3.999) de memoria, nuestro programa debe ser cargado por el «loader» a partir de la dirección 4.000 (inclusive) o superiores.

Supongamos que se ha efectuado así:

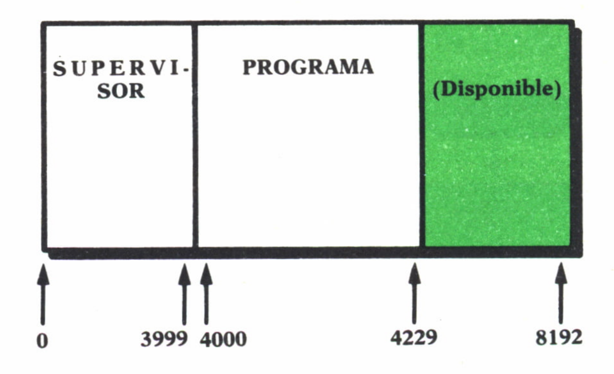


Fig. 8.

Ahora el «loader» transfiere control a la primera instrucción de nuestro programa (posición 4.000). Si la primera instrucción es, por ejemplo, un salto incondicional a la que está en la posición 208, esto sería erróneo, pues lo que debería hacer es saltar a la posición 4.208. El error se produce porque el ensamblador ha calculado sus direcciones a partir de la posición 0 y el programa lo hemos cargado en otra (en la 4.000).

La solución más simple a este problema consiste en hacer que el ensamblador calcule las direcciones empezando en la posición 4.000, que es donde sabemos que puede cargarse el programa, puesto que la 4.000 es la primera posición de memoria libre después del *supervisor* (cuyo tamaño es conocido por el programador). Esto puede conseguirse con una seudoinstrucción en ensamblador, cuyo operando contenga la dirección a partir de la cual queremos que el ensamblador calcule las direcciones. De esta manera, el programa objeto que se obtendrá funcionaría correctamente una vez cargado por el «loader» en la posición 4.000.

La solución indicada no es totalmente satisfactoria, porque si nos vemos obligados a incluir nuevas funciones y rutinas en el *supervisor* y éste aumenta de tamaño, nos veríamos en la necesidad de reensamblar todos los programas de nuestra instalación, cambiando la seudoinstrucción en

todos. Esto nos indica la conveniencia de que el desajuste entre las direcciones calculadas por el ensamblador y aquéllas donde realmente se ha cargado el programa, se resuelva en el momento de cargar el programa en memoria. Para ello hay que:

1. Hacer que el ensamblador indique (mediante una señal o creando una tabla) *qué direcciones* del código máquina deben ser reajustadas si el programa de carga en direcciones distintas a las que se han ensamblado.
2. Hacer que el «loader», una vez cargado el programa en memoria y antes de empezar su ejecución, repase y recalcule estas direcciones indicadas por el ensamblador.

Este problema es importante por dos razones, además de lo expuesto en las líneas precedentes:

1. La programación modular consiste en que un programa emplee rutinas ensambladas separadamente, y que, por tanto, en el momento de ensamblarlas no se sabe en qué posición de memoria se van a ejecutar.
2. Cuando se emplea multiprogramación (es decir, varios programas coexisten a la vez en la memoria) se aprovecha el primer hueco que queda disponible en la memoria para cargar nuestro programa, siendo imposible predecir para el programador, mientras está escribiendo su programa, en qué posición de memoria se ejecutará.



MONTADOR DE ENLACE (LINKAGE EDITOR)

Se ha mencionado la programacion modular en el párrafo anterior. Básicamente consiste en descomponer un programa en módulos o subprogramas, cada uno de los cuales se programa y ensambla separadamente. Cuando todos están a punto, se unen para formar el programa final. El programa que se encarga de unir los distintos módulos objeto (es decir, ya compilados y en código de máquina) para formar el programa o módulo cargable final, se llama montador de enlaces «linkage editor».

Podemos indicar algunas ventajas de programar modularmente:

— Cuando hay un fallo en un subprograma, sólo éste debe recompilarse.

— Los subprogramas pueden colocarse en librerías donde futuros programadores pueden utilizarlos (por ejemplo: subprogramas que realicen funciones de uso general, como extraer raíces cuadradas, calcular intereses de un capital dado, etc.).

— Es más fácil programar muchos subprogramas pequeños que uno grande.

— Los módulos pueden ser programados por programadores distintos simultáneamente, acortándose el tiempo total de programación.

— El tiempo total de pruebas se acorta si se prueban los módulos independientemente.

— El mantenimiento del programa se simplifica (por ejemplo: si hay un módulo que calcula las primas por horas extra en un programa de nóminas, y se modifica el procedimiento de pago de primas, sólo hay que modificar este módulo).

El programa montador de enlaces, como los compiladores, no forma parte del *supervisor* y, por tanto, no reside permanentemente en memoria. Es un programa auxiliar, englobado en el conjunto de programas que forman parte del sistema operativo.

Los datos de entrada para el «linkage editor» son los módulos objeto (es decir, en código máquina). El «linkage editor» los lee y produce como salida un programa cargable, en soporte legible por el ordenador (disco, cinta, fichas, papel). Este módulo cargable es cargado por el «loader» cuando lo queremos ejecutar.

Por tanto, entre la salida del compilador y el «loader» hemos introducido un paso intermedio, el «linkage editor», para resolver el problema de fundir varios módulos en uno.

¿Por qué es necesario este paso adicional? Vamos a intentar explicarlo a grandes rasgos.

Supongamos un programa A que necesita calcular raíces cuadradas y que hay una subrutina, ya compilada por otro programador y disponible, que calcula raíces cuadradas, y que se llama SQRT. El programa A, en algún momento, dará una orden de saltar a la rutina SQRT.



Fig. 9. Programa A.

Es evidente que, al no estar el símbolo SQRT definido en el programa fuente A, el ensamblador no sabe qué dirección de máquina corresponde a este símbolo. Entonces lo deja a ceros, y lo incluye en una tabla, que se

perfora formando parte del módulo objeto. Esta tabla reúne todos los símbolos del programa fuente "no resueltos" por el ensamblador (como el SQRT).

Cuando el «linkage editor» lee los módulos objeto de A y SQRT, resuelve las direcciones indicadas en la tabla anterior, insertando en la instrucción SAI SQRT del módulo A la dirección de la subrutina SQRT.

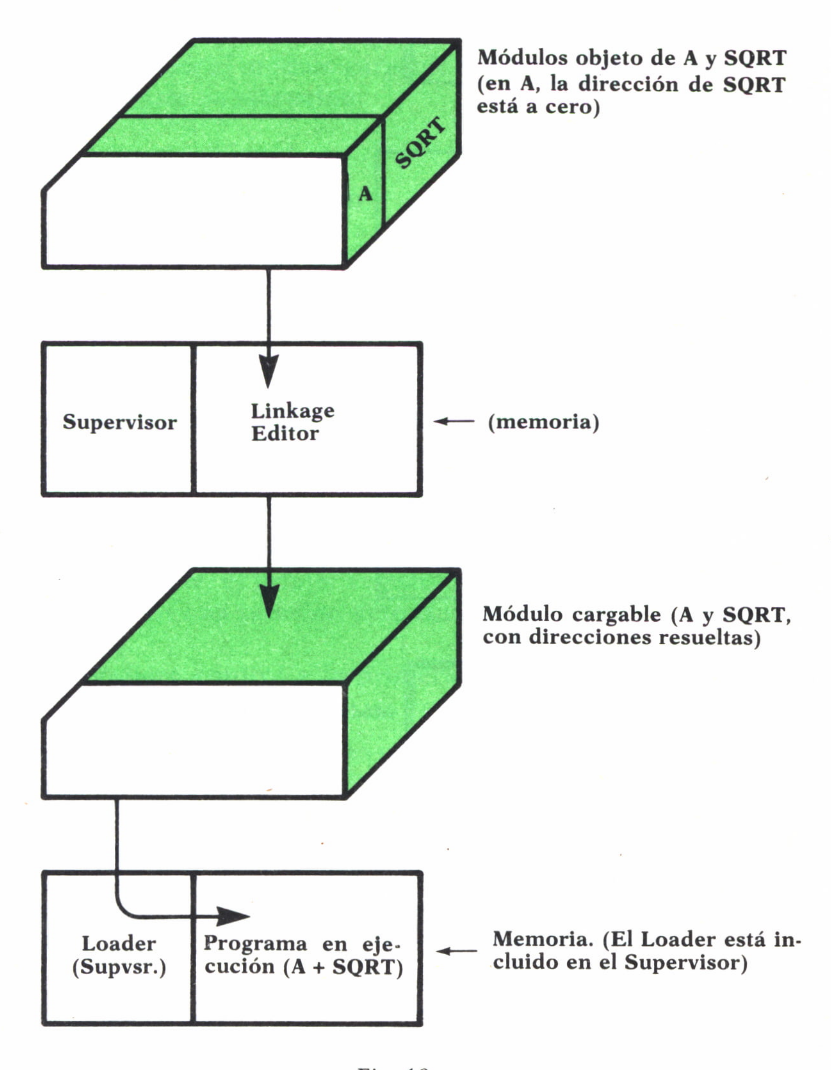


Fig. 10.

En resumen, las distintas etapas por las que pasa un programa pueden esquematizarse así:

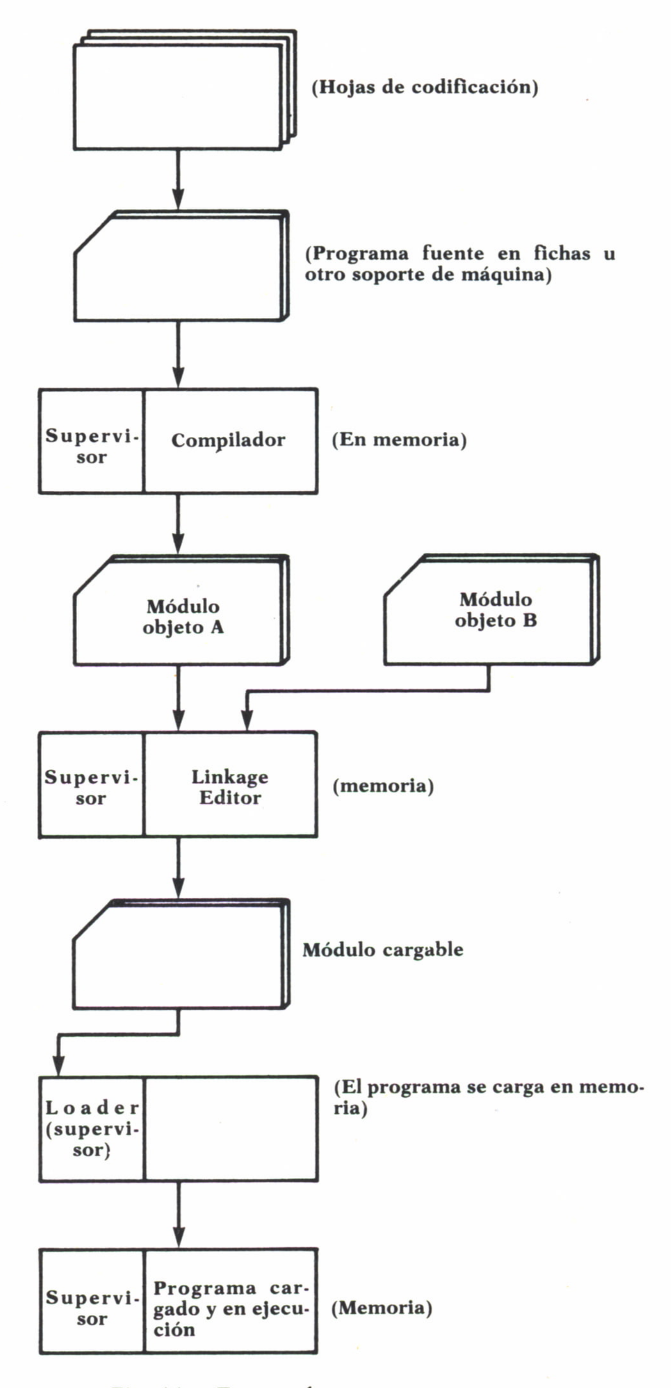


Fig. 11. Etapas de un programa.

Los módulos cargables obtenidos como salida del «linkage editor» se suelen grabar en disco, en un fichero o librería donde cada uno de ellos está identificado con un nombre, elegido por el programador. Cuando hay que ejecutar un programa, el operador facilita al programa *supervisor*, mediante mensajes a través del teclado de la consola, los datos siguientes: nombre del fichero o librería donde está el módulo cargable que se quiere ejecutar, nombre con el que éste está identificado dentro de la librería y posición de memoria donde se quiere cargar el programa. El «loader» es responsable de leer el módulo cargable desde el disco, introducirlo en memoria y comenzar su ejecución.



CARGA INICIAL

Hemos visto que los programas son cargados en memoria por el «loader», que es una rutina del *supervisor*. Pero ¿quién carga en memoria al *supervisor*?

La solución más común a este problema sigue las directrices que, sin profundizar, vamos a exponer a continuación.

El ordenador, en la mayor parte de las instalaciones, dispone de una tecla, normalmente llamada tecla de **LOAD** (carga), en el panel. Cuando el operador aprieta esta tecla, el ordenador por microprograma o cableado hardware, hace lo siguiente:

1. Lee un bloque (una ficha perforada, un registro perforado en cinta o, lo más frecuente, el primer registro grabado en una unidad determinada de discos), en una posición determinada de memoria (normalmente al final de ésta).
2. Transfiere control a la primera posición de memoria del bloque leído.

Este bloque que se ha leído es una pequeña rutina que lee otros dos bloques más en las posiciones contiguas. El conjunto de todos estos bloques situados en las últimas posiciones de memoria forman un programa «loader» restringido (no puede hacer reubicación de los programas que carga), que lee ya el *supervisor* completo, y lo introduce en memoria. A partir de aquí el *supervisor* empieza a trabajar como se ha descrito en líneas precedentes.



RESUMEN

— El «loader» es una rutina del *supervisor* cuya función es cargar los programas en memoria y transferirles control.

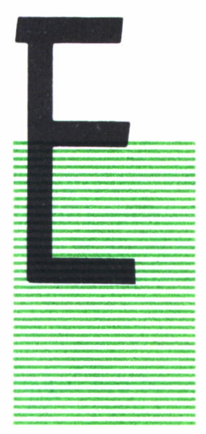
— El «loader» debe ser capaz de reajustar las direcciones de las ins-

trucciones de los programas que carga en memoria, de tal modo que éstos funcionen correctamente sea cual sea la posición origen de la carga.

— El «linkage editor» tiene por objeto resolver las direcciones de referencia a subrutinas compiladas separadamente.

— El «linkage editor» es un programa auxiliar dentro del conjunto del sistema operativo; no forma parte del *supervisor* y, por tanto, no está en memoria cargado permanentemente, sino cuando el operador lo decida.

INTERRUPCIONES 3



EN este capítulo vamos a intentar describir el mecanismo de las *interrupciones*.

Un programa *supervisor* está formado por rutinas que cabe agrupar en dos categorías:

1. *Programa de control.*

Es la parte del *supervisor* que decide a qué rutina o programa se ha de transferir control en un momento dado. Es, metafóricamente, el director de orquesta de todos los programas y rutinas (incluidas las del propio *supervisor* y las del usuario) que hay cargados en memoria.

2. *Rutinas de servicio.*

Son las restantes rutinas del *supervisor*, por ejemplo, el «loader» ya citado.

En el capítulo 1 citábamos que cuando un programa se está ejecutando, coexistiendo en memoria con el *supervisor*, hay entre ambos unas frecuentes transferencias de control de datos. Cada vez que el programa pasa control al *supervisor* para pedirle un servicio (por ejemplo, que realice una operación de E/S, que mande un mensaje a la consola del operador, etc.) no lo hace mediante una instrucción de salto, sino mediante una instrucción especial de llamada al supervisor (SVC=Supervisor Call). Esta instrucción provoca una transferencia de control mediante un mecanismo de hardware que se llama interrupción. El núcleo de la rutina de control del *supervisor* es el tratamiento de las interrupciones.

El mecanismo de interrupciones funciona básicamente igual en casi todos los ordenadores.

Vamos a empezar este capítulo describiendo un modelo sencillo de hardware con interrupciones. Esto nos dará base para, posteriormente, describir un programa de control simplificado.



INTERRUPCIONES

Cuando la unidad de control del ordenador ejecuta un programa cargado en memoria, lo hace una instrucción tras otra, secuencialmente. Supongamos que durante la ejecución del programa el operador decide cancelarlo (por ejemplo, porque se da cuenta de que ha preparado mal los datos de entrada del programa). Entonces, el operador debe teclear un mensaje en la consola para indicar al *supervisor* que suspenda la ejecución del programa. Esto implica que el *supervisor* debe ejecutar una instrucción de leer desde la máquina de escribir de la consola para aceptar el mensaje tecleado por el operador.

¿Cómo puede dar el *supervisor* esta instrucción si en este momento la unidad de control está ejecutando el programa?

Esta es una situación típica que resuelve el mecanismo de interrupciones. En nuestro ejemplo, el operador, una vez tecleado su mensaje (mientras el programa sigue ejecutándose), pulsaría una tecla especial de la consola, que llamaremos tecla INPUT. Al apretar esta tecla se produce una interrupción, que consiste en:

1. La unidad de control deja de ejecutar el programa.
2. La unidad de control pasa a ejecutar la rutina de control del *supervisor.*

Entonces, la rutina de control del *supervisor* analizará la causa de la interrupción (puede haber muchas), detectará que ha sido el operador al apretar la tecla INPUT, y dará una instrucción de leer el mensaje de la consola.



MECANISMO DE INTERRUPCIONES

Para poder realizar las interrupciones hay que dotar al hardware de cierto mecanismo que permita romper la secuencia normal de ejecución, que llamaremos **«bit de interrupción»** y **«máscara de interrupciones»,** que normalmente están a cero. El «bit de interrupción» es un registro de un solo bit, que se pone a 1 (ON) cada vez que surge una interrupción (por ejemplo, cuando el operador aprieta la tecla INPUT). El registro «máscara de interrupciones» consta de varios bits, uno por cada tipo de interrupción que puede producirse.

Así, cuando el operador aprieta INPUT, se pone a 1 el bit correspondiente a esta interrupción en la «máscara de interrupciones», además del «bit de interrupciones».



Fig. 12. Organigrama del mecanismo de interrupciones.

Por otra parte, recordemos que una instrucción típica es ejecutada por la unidad de control en dos fases:

1. Fase «fetch»: extraer la instrucción de memoria e introducirla en los registros internos de la unidad de control.
2. Fase «execute»: encaminar los operandos, decodificar el código de operación, ejecutar la instrucción y poner en el contador de instrucciones la dirección de la siguiente.

Cuando queremos dotar al ordenador de interrupciones, hay que completar estas fases con otra en la que la unidad de control, despues de la fase segunda («execute»), examina el «bit de interrupciones». Si está en OFF, ejecuta la instrucción siguiente. Si está en ON, debe bifurcar a la rutina de control. Si suponemos que ésta está cargada en memoria en la posición 1, la unidad de control pondrá la dirección 1 en el contador de instrucciones, para que la siguiente instrucción a ejecutar sea la primera de la rutina de control.

Por otra parte, el programa cuya ejecución se ha interrumpido deberá ser reanudado una vez que la rutina de control haya terminado de tratar la interrupción. Para ello, hay que guardar en algún sitio de memoria (en la posición 0, por ejemplo) la dirección de la instrucción siguiente a aquella en que se ha interrumpido.

Lo recogido en este párrafo puede ponerse en forma de organigrama (figura 12).



TIPOS DE INTERRUPCIONES

En la «máscara de interrupciones» cada bit representa un tipo de interrupciones. Las interrupciones podemos clasificarlas en las categorías siguientes:

1. **Interrupciones de entrada/salida.** Cada bit de la «máscara de interrupciones» representa un dispositivo de E/S. Se produce una interrupción de E/S cuando se acaba una operación de E/S.
2. **Interrupciones de programa.** Reservamos un bit en la «máscara de interrupciones» para este tipo (bit 10 de la máscara). Cuando un programa solicita un servicio del *supervisor,* lo hace mediante una instrucción SVC, como ya hemos indicado. Cuando la unidad de control ejecuta la instrucción SVC, lo que hacer es poner ON este bit de la «máscara de interrupciones» y el «bit de interrupciones». Esto provoca automáticamente una interrupción, es decir, una transferencia de control a la rutina del *supervisor,* el cual, tras analizar la interrupción, proporcionará al programa el servicio solicitado y le devolverá control.
3. **Interrupciones de error de programa.** Cuando la unidad de control detecta un error en la ejecución de una instrucción, provoca una in-

terrupción de este tipo. En la «máscara de interrupciones» reservamos tres bits para estas interrupciones, que indican:

— Código de operación inválido (bit 11).

— Operando inválido (bit 12).

— Overflow aritmético (bit 13).

4. **Interrupción del operador.** Causadas por pulsar teclas de la consola con ciertas funciones específicas. Reservaremos tres bits en la «máscara de interrupciones»:

— Tecla INPUT. La usará el operador cuando quiera comunicar algún mensaje al *supervisor* (bit 14).

— Tecla STOP. Cuando quiera parar la ejecución de un programa (por ejemplo, para hacer un «volcado» de memoria) (bit 15).

— Tecla START. Cuando se quiera reanudar la ejecución de un programa detenido con STOP (bit 16).

En resumen, si suponemos que tenemos 10 dispositivos de E/S, la «máscara de interrupciones» será un registro de 17 bits:

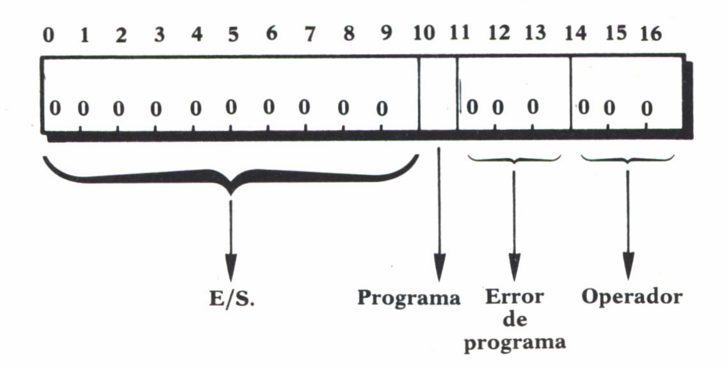


Fig. 13. Máscara de interrupciones.



RUTINA DE CONTROL DEL SUPERVISOR

Su función principal es analizar las interrupciones y decidir qué rutina debe recibir el control.

Cuando surge una interrupción, se pone en ON el bit correspondiente de la «máscara de interrupciones» y el «bit de interrupciones». Cuando la unidad de control, como consecuencia de una interrupción, empieza a ejecutar la rutina de control del supervisor, que está en la posición 1 de memoria, lo primero que debe hacer esta rutina de control es averiguar la causa de la interrupción. Esto lo hace explorando los bits de la «máscara de interrupciones» de izquierda a derecha, hasta encontrar uno en ON. En-

tonces salta a la rutina que trate este tipo de interrupción. Un esquema en forma de organigrama de la rutina de control sería:

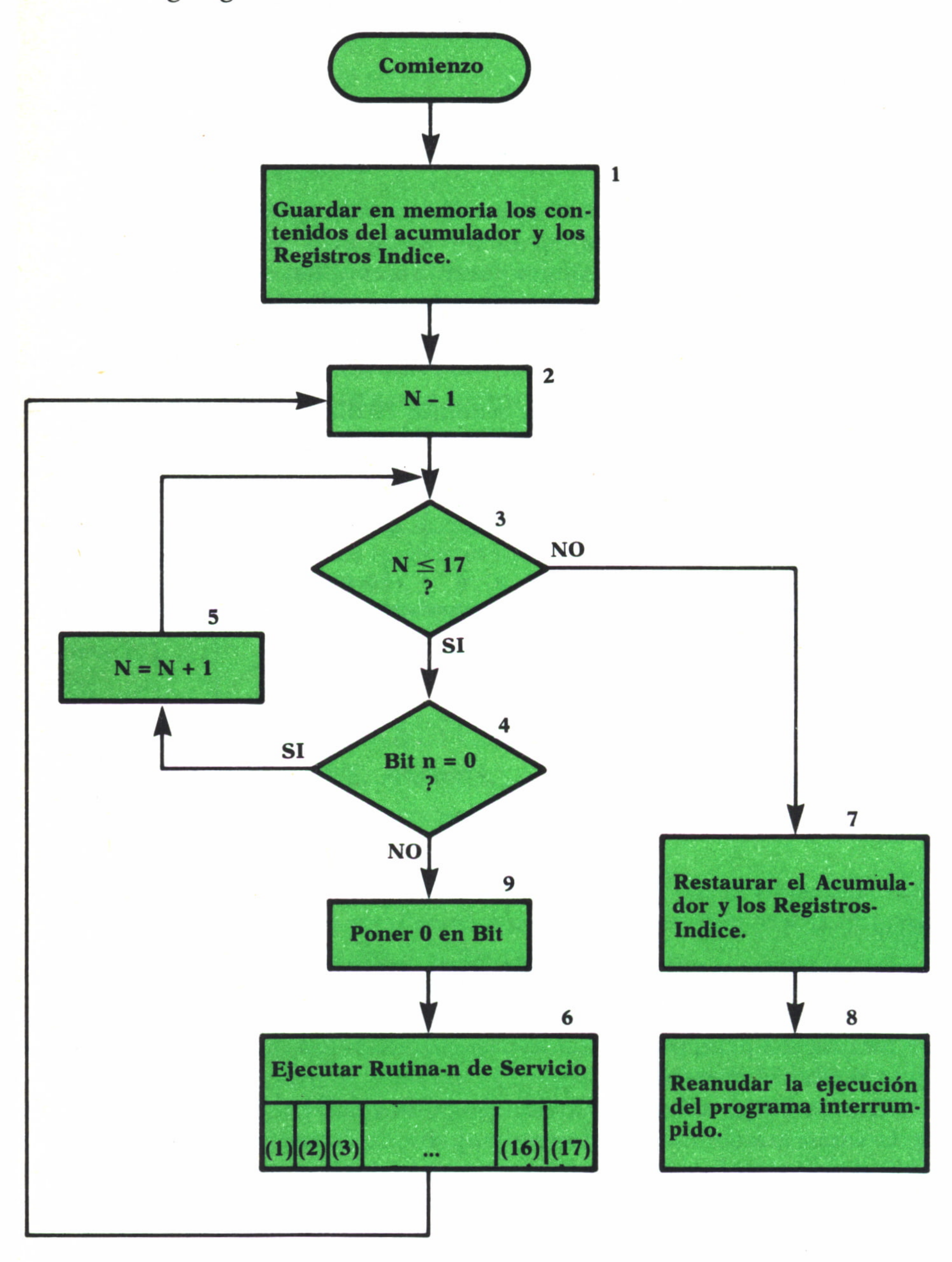


Fig. 14. Organigrama de la rutina de control.

Lo primero que hace (bloque 1 del organigrama anterior) la rutina de control es salvar el contenido de los registros índice y del acumulador. Esto es necesario para poder restaurar el estado del programa interrumpido cuando se termine de tratar la interrupción (ver bloque 7). El contador de instrucciones no es salvado en el bloque 1, porque lo salva el hardware automáticamente. Una vez terminado el tratamiento de la interrupción, la rutina de control restaura los registros índice, el acumulador y el contador de instrucciones (bloques 7 y 8), con lo que se reanuda la ejecución del programa interrumpido.

Los bloques 2, 3, 4 y 5 del organigrama exploran de izquierda a derecha los 17 bits de la «máscara de interrupciones», hasta encontrar uno en ON. Entonces se transfiere control a una rutina de servicio que trate la interrupción. Habrá 17 rutinas de éstas (una por cada bit de la «máscara de interrupciones»). Esto se indica de una forma general en el bloque 6.

Una vez que la rutina de servicio elegida termina de ejecutarse, se recomienda la búsqueda del primer bit en ON de la «máscara de interrupciones». Esto se hace así porque, durante la ejecución del *supervisor*, pueden surgir causas de interrupción (por ejemplo, el operador aprieta la tecla INPUT). En este caso, no se produce interrupción, es decir, la unidad de control no rompe su secuencia normal de ejecución, pero sí se pone ON en el bit de la «máscara de interrupciones» correspondiente a la interrupción surgida. Por tanto, la unidad de control debe distinguir cuándo está ejecutando instrucciones del *supervisor* y cuándo del usuario. En el primer caso, no funcionará el mecanismo de interrupciones y en el segundo sí.

En resumen, durante la ejecución del bloque 6 pueden ponerse ON algunos bits de la «máscara de interrupciones», por lo que hay que volver a explorar ésta.



RESUMEN DEL CAPITULO

— Las transferencias de control al programa supervisor se efectúan por medio de *interrupciones*.

— Cuando se produce una interrupción, la unidad de control deja de ejecutar el programa que estuviera en ese momento ejecutando, y pasa a ejecutar el supervisor (la rutina de control). Este devuelve control al programa interrumpido cuando termina de tratar la interrupción.

— El hardware debe estar provisto de los registros y mecanismos necesarios.

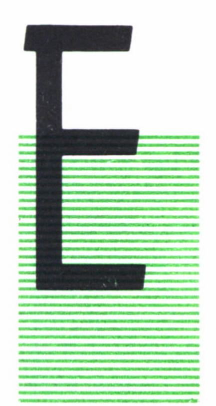
— El núcleo del programa *supervisor* es la *rutina de control,* cuya responsabilidad es analizar las interrupciones y, dependiendo de su causa,

transferir control a la rutina de servicio apropiada. El supervisor está formado, pues, por la rutina de control y las de servicio.

— Las interrupciones se clasifican en cuatro tipos:

  — De E/S (cuando acaba una operación).
  — De programa (cuando se ejecuta la instrucción SVC).
  — De errores de programa.
  — Del operador.

OPERACIONES DE ENTRADA/SALIDA 4



EN capítulos anteriores se ha comentado que uno de los servicios que el *supervisor* realiza, a petición de los programas, es la entrada/salida (E/S), y que toda operación de E/S provoca una interrupción de este tipo.

La razón para incluir toda la complejidad de programación de la E/S en el *supervisor* es, como siempre, liberar al programador del máximo posible de trabajo; las rutinas que efectúan la E/S se prestan muy bien a esto, dada su generalidad de uso (prácticamente todos los programas necesitan ejecutar operaciones de E/S). Por otra parte, su programación es bastante compleja debida, fundamentalmente, a los siguientes factores:

1. Un programa puede manejar un elevado número de dispositivos de E/S (por ejemplo, una lectora de fichas, una perforadora, una impresora, una cinta de entrada con datos maestros, un fichero en disco con datos consultables, etc.).
2. La casuística de los posibles errores a detectar, y recuperar si es posible, es muy variada y además diferente según el tipo de dispositivo (por ejemplo, perforaciones erróneas en una cinta de papel o en una ficha, cintas magnéticas deterioradas, cabezas magnéticas de un disco estropeadas, cuando un carrete de cinta magnética se acaba hay que avisar al operador para que monte otro, etc.).
3. La optimización de las operaciones de E/S exige una programación sofisticada y engorrosa (por ejemplo, manejo de múltiples «buffers»).
4. Cada dispositivo admite distintas formas de almacenar y organizar los datos, por lo que el acceso a éstos dependerá del tipo de dispositivo (por ejemplo, en cinta los registros se graban secuencialmente; en cambio, en disco pueden grabarse secuencialmente, al azar, o con índices, etcétera).

Todo esto hace de las rutinas de E/S una parte muy importante y voluminosa del *supervisor*.



CONCEPTO BASICO DE CANAL

Supongamos un programa que lea fichas y haga ciertos cálculos con cada una de las fichas que lee, con un organigrama como el siguiente:

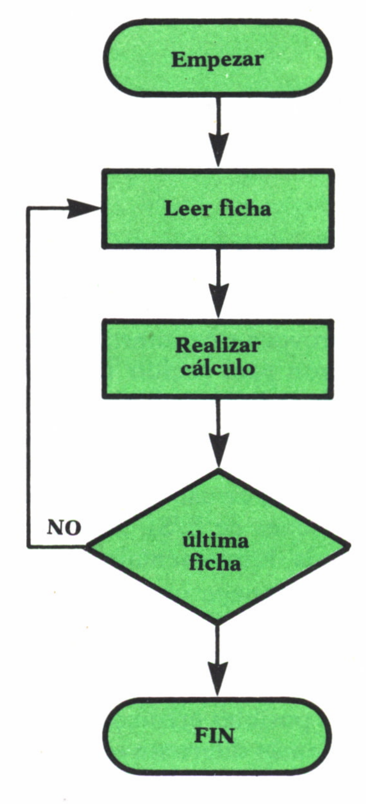


Fig. 15. Programa.

El ciclo leer-calcular, a lo largo del tiempo, podría representarse así:

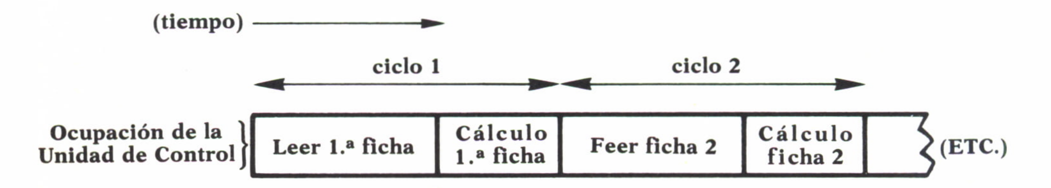


Fig. 16. Ciclo leer-calcular.

En la práctica este ciclo está muy descompensado, es decir, el tiempo de lectura suele ser mucho mayor que el de cálculo, debido a que el tiempo de ejecución de instrucción de E/S depende de la velocidad del dispositivo de E/S, que normalmente es mucho más lento que la Unidad Central (debido a que el dispositivo de E/S tiene órganos mecánicos y la unidad de control sólo elementos electrónicos). Durante toda la lectura del ciclo leer-calcular anterior, la unidad de control ha ejecutado una sola instrucción (leer), cuando en ese tiempo podría haber realizado mucho más trabajo.

Este problema se resuelve incorporando al ordenador un dispositivo llamado **canal.** Este es como una pequeña unidad central, especializada sólo en ejecutar intrucciones de E/S, con sus propios registros internos (sin acumulador). Entonces, cuando la unidad de control llega a la instrucción de leer, lo que hace es transferir esta instrucción a los registros internos del canal y dar a éste una orden para que ejecute la instrucción. A partir de este momento, la unidad de control, por lo que a ella respecta, considera ejecutada la instrucción de leer, y pasa a ejecutar la siguiente en el programa. Mientras tanto, el canal se encarga de leer la ficha, transfiriendo los datos de ésta a memoria conforme la ficha va pasando por las escobillas del dispositivo de lectura. En resumen, el canal se encarga de la E/S liberando a la unidad de control, que puede seguir trabajando mientras tanto.

Esto supone una mejora muy importante en el rendimiento del ordenador.

Para fijar ideas, veamos algunas cifras típicas sobre velocidades de E/S y proceso (naturalmente, estas cifras pueden variar mucho, según los tipos y modelos de máquinas; los valores aquí dados son para fijar órdenes de magnitud):

Dispositivo	**Velocidad**
Lectora de fichas	600 fichas/minuto
Impresora	1000 líneas/minuto
Máquina escribir de consola	600 caracteres/minuto
Cinta magnética	3.000.000 de caracteres/minuto
Disco magnético	5.000.000 de caracteres/minuto

Unidad central	**Velocidad**
Instrucciones de sumar	10.000.000 de sumas/minuto
Acceso a memoria	60.000.000 de palabras/minuto

Apliquemos estas cifras al ciclo leer-calcular del programa de la figura

15, suponiendo que por cada ficha se realizan 600 sumas y 400 accesos a memoria:

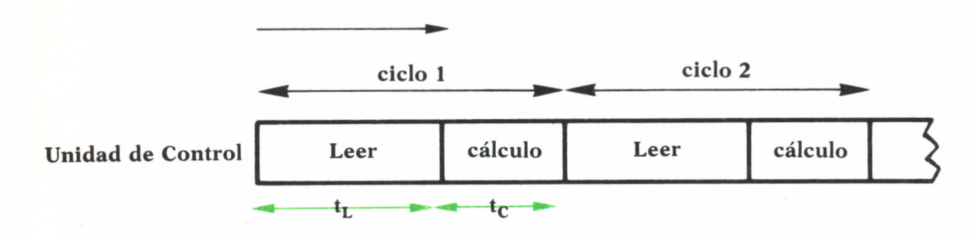


Fig. 17. Ciclo leer-calcular sin canal.

En la figura 17 tendremos:

t_L = 100 ms.

t_C = 4 ms.

CICLO = 104 ms.; 100/104 = 96%

Es decir, si no se utiliza un canal para simultanear la E/S y el proceso, el 96% del tiempo la unidad de control está sin realizar trabajo útil.

En cambio, si el ordenador dispone de un canal para realizar la E/S, el esquema de tiempos podría ser el de la figura 18.

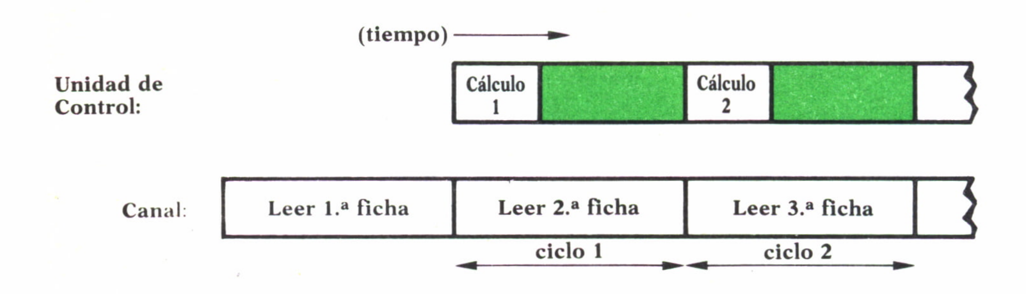


Fig. 18. Ciclo leer-calcular con canal.

En la figura 18 vemos que, como la unidad de control y el canal pueden trabajar simultáneamente, podemos hacer el programa de tal modo que mientras el canal lee la ficha n-sima, la unidad de control procesa la (n-1)sima (habría que modificar el organigrama de la figura 15).

Tendríamos:

$$CICLO = t_L = 100 \text{ ms.}$$

En la práctica, un programa maneja más de un dispositivo de E/S. Supongamos que un programa lee una ficha, hace algunos cálculos con ella, los imprime y repite este ciclo por cada ficha que lee. Supongamos también que cada dispositivo de E/S está conectado a un canal diferente, es decir, que trabaja simultáneamente con los otros y con la unidad de control.

El esquema de tiempos del programa anterior, sin emplear canales (es decir, sin solapar tiempos) sería el de la figura 19.

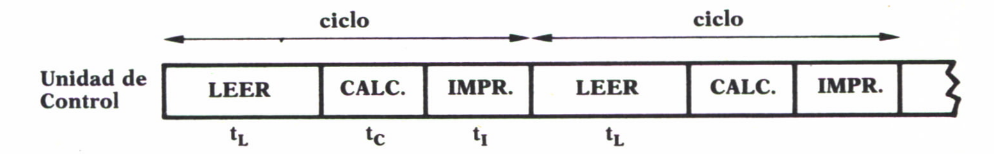


Fig. 19. Esquema de tiempos sin canal.

Los tiempos serían:

t_L = 100 ms.

t_C = 4 ms.

t_I = 60 ms.

CICLO = 164 ms.

Con solapamiento, el esquema sería el de la figura 20.

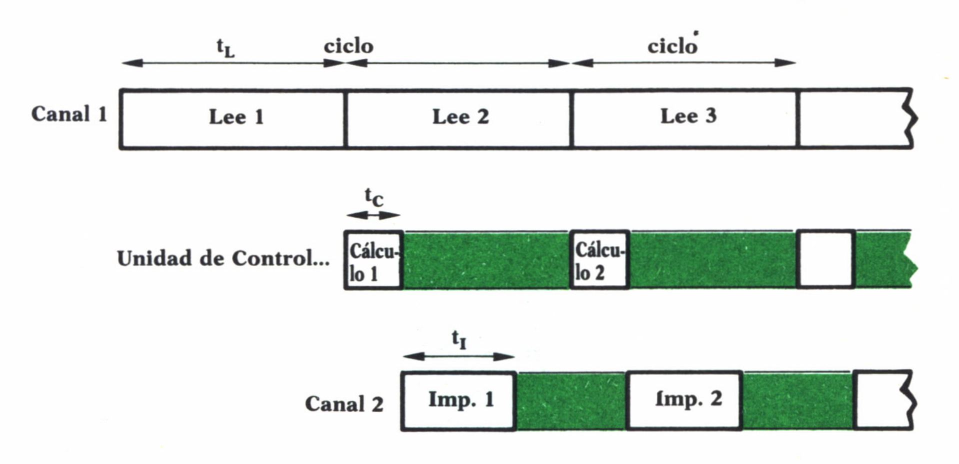


Fig. 20. Esquema de tiempos con canal.

El ciclo sería de 100 ms.
Si el programa tiene que leer 10000 fichas, su duración será:

(sin canales): T = 10000 × 164 ms = 1640 seg = 27 min.
(con canales): T = 10000 × 100 ms = 16 min.

Se ha ganado un 40% de tiempo en este programa gracias a la utilización de canales.



MODALIDADES DE OPERACION

Un canal puede realizar la transferencia de datos entre los dispositivos de E/S y la memoria principal de tres formas:

Modalidad a ráfagas

En esta modalidad el canal transmite los datos como un bloque continuo. Para dispositivos tales como cintas y discos que trabajan con una velocidad de transmisión alta, los datos deben ser transmitidos utilizando esta modalidad de transmisión, que es forzada por el dispositivo, no por elección del programador.

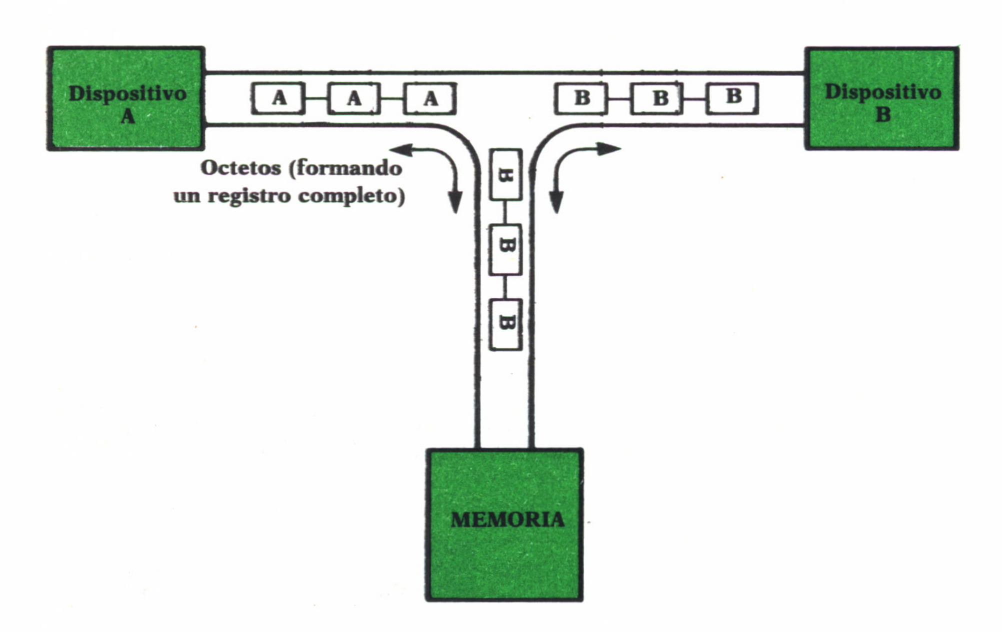


Fig. 21. Modalidad de ráfagas.

Modalidad de octetos

En esta modalidad un solo camino del canal puede ser compartido por varios dispositivos de baja velocidad, tales como lectoras de fichas, impresoras, etc.

Estos dispositivos transmiten la información con suficiente lentitud como para que entre la transmisión de un octeto y otro, el canal tenga tiempos muertos que puede utilizar para la transmisión de octetos con otro dispositivo. De esta forma, varios dispositivos de lenta velocidad pueden trabajar concurrentemente con un solo canal. Los octetos de datos que provienen de varios dispositivos, viajan intercalados, pero el canal los dirige correctamente al dispositivo de E/S que corresponda o la posición de memoria deseada.

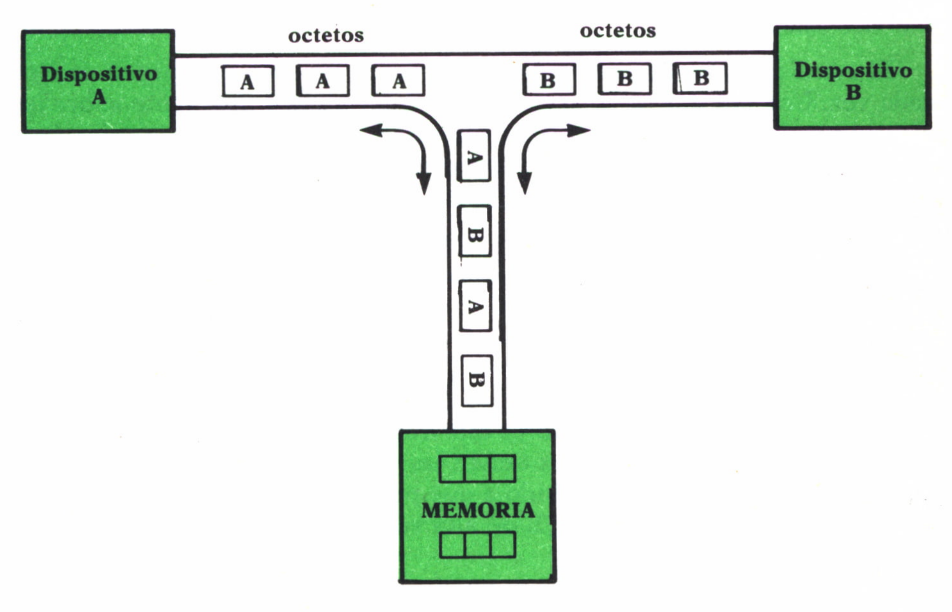


Fig. 22. Modalidad de octetos.

Intercalación de bloques

Esta modalidad es similar a la anterior, pero en vez de ir intercalando octetos que provienen de distintos dispositivos, intercala bloques completos de datos de diferentes dispositivos.



TIPOS DE CANALES

Mencionaremos los tipos de canales mas conocidos.

Canales multiplexores de octetos

Este es un canal diseñado para trabajar simultáneamente con varios dispositivos de E/S. Esto es, varios dispositivos de E/S pueden estar transmitiendo datos al mismo tiempo.

El *canal multiplexor de octetos* trabaja en diferente modalidad, según trabaje con dispositivos de baja velocidad o con dispositivos de alta velocidad. Con los primeros trabaja en *modalidad de octetos* y con los segundos en *modalidad de ráfagas.*

Canales selectores

El *canal selector* transmite datos a/o desde un solo dispositivo de E/S cada vez. Una vez que se selecciona una determinada operación sobre un dispositivo de los que el canal tiene conectados, no podrá iniciar otra, hasta que ésta haya concluido completamente. Los canales selectores trabajan solamente en *modalidad de ráfagas.*

Un canal selector puede operar con dispositivos de baja y alta velocidad, pero su forma de trabajo, *a ráfagas,* lo hacen especialmente adecuado para trabajar con estos últimos.

Canales multiplexores de bloques

Estos son canales multiplexores que intercalan bloques de datos en vez de octetos. Este tipo de canales tienen las ventajas de los dos tipos de canales vistos anteriormente, debido a que pueden operar concurrentemente con varios dispositivos de E/S de alta velocidad (en un solo camino de datos).

El *canal multiplexor de bloques* puede operar en *modalidad selectora* o en *modalidad de bloques.*



RESUMEN

— Una responsabilidad fundamental del *supervisor* es el lanzamiento de las operaciones de E/S, y el tratamiento de las interrupciones de E/S originadas cuando acaban.

— El trabajo de programación de E/S se ve muy simplificado gracias al soporte proporcionado por las rutinas del *supervisor*.

— El *supervisor* se encarga de realizar la operación de E/S, y tratar los posibles errores que se puedan producir.

— Los canales son pequeñas unidades centrales especializadas en ejecutar intrucciones de E/S.

MULTIPROGRAMACION



HEMOS visto que la operación concurrente de los canales permite mejorar el rendimiento del ordenador. Así, por ejemplo, en las figuras 19 y 20 del capítulo 4 vimos que la duración del programa considerado (en el caso de leer 10000 fichas), se acortaba desde veintisiete minutos, sin operación concurrente de canales, a dieciséis minutos con ella. Esto representaba una notable ganancia de tiempo del 40%.

Sin embargo, la utilización de la unidad de control puede mejorarse todavía más. En efecto, de acuerdo con la figura 19 del capítulo 4 vemos que el porcentaje de tiempo que la unidad de control está realizando trabajo útil es:

$$(\text{Uso de la U.C.}) = t_C/\text{- CICLO} = t_C/t_L + t_C + t_I = 4/164 = 2{,}4\ \%$$

Veamos ahora la utilización que corresponde a la figura 20 del capítulo 4.

$$(\text{Uso de la U.C.}) = t_{C}\text{-/CICLO} = t_C/t_L = 4/100 = 4\ \%$$

En resumen, hemos mejorado la utilización de la unidad de control desde un 2,4 a un 4%, pero sigue siendo una cifra muy baja, pues el 96% del tiempo está sin realizar trabajo útil. Esta infrautilización es gravemente antieconómica, por ser la unidad de control uno de los componentes más caros del ordenador.

La pregunta evidente ante esta situación es: ¿podríamos emplear para otros programas la capacidad de proceso que le sobra a la unidad de control?

Para conseguir esto se emplea la técnica de **multiprogramación.**

En este capítulo vamos a exponer en qué consiste la multiprogramación, y algunos de los problemas que se plantean para conseguirla.



CONCEPTOS BASICOS DE MULTIPROGRAMACION

Hasta ahora, siempre hemos considerado cargado en memoria, en un momento dado, un solo programa del usuario, a la vez que el *supervisor:*

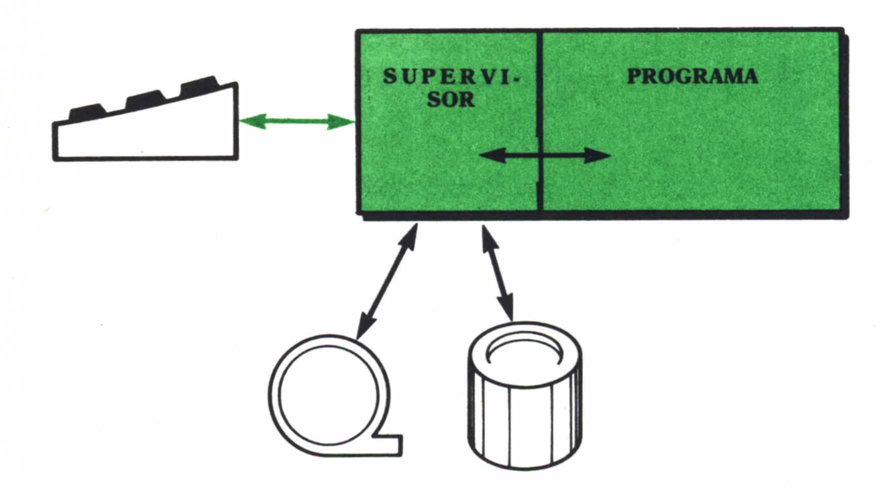


Fig. 23.

Entonces, se plantea el problema que comentábamos en líneas anteriores: como el programa tiene tiempos muertos, la unidad de control está infrautilizada.

La causa de que un programa tenga tiempos muertos puede ser, fundamentalmente, una de las dos siguientes:

1. El programa necesita esperar a que se acabe una operación de E/S antes de seguir ejecutándose.

2. El programa necesita esperar a que el operador haga alguna operación (por ejemplo, si el programa necesita imprimir, y no hay papel en la impresora, hay que esperar a que el operador ponga el papel).

La multiprogramación consiste en tener cargado en memoria más de un programa del usuario, junto con el *supervisor:*

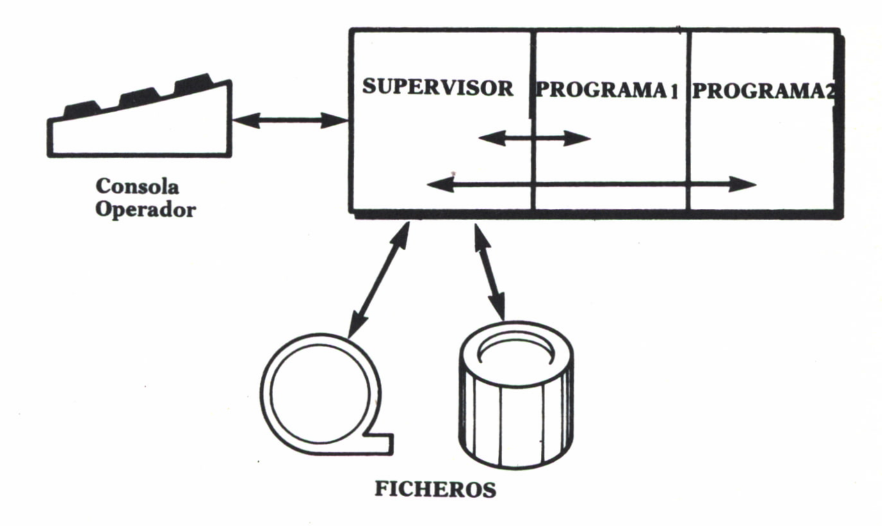


Fig. 24.

Supongamos que en la figura anterior, en un momento dado, la unidad de control está ejecutando el programa 1. Si éste necesita esperar a que ocurra algún suceso antes de continuar (por ejemplo, una intervención del operador), el *supervisor* detecta esta situación y transfiere control al programa 2. Es decir, la unidad de control, como siempre, ejecuta las instrucciones una a una (nunca más de una a la vez), pero aprovecha las esperas obligadas de un programa para ejecutar instrucciones del otro. En definitiva, utilizamos más la unidad de control.

La operativa del sistema sería:

1. Al comienzo del día, el operador hace IPL y carga el *supervisor* en memoria.

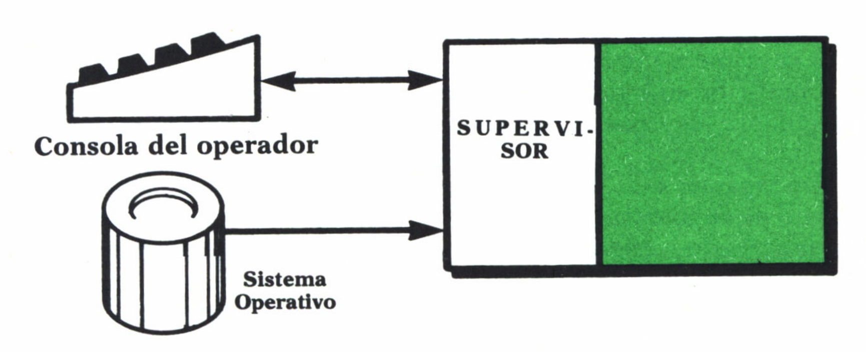


Fig. 25. Carga del supervisor en memoria.

2. El operador pide al *supervisor* (a través de un mensaje que teclea en el teclado de la consola) que cargue el programa 1. Para ello, el operador proporciona al *supervisor los datos siguientes:*

a) Nombre de la librería de disco donde se encuentra el módulo cargable que se quiere ejecutar.

b) Nombre que identifica al módulo cargable dentro de su librería.

c) En qué posición de memoria queremos que se cargue el módulo.

d) Qué dispositivos de E/S y qué ficheros maneja el programa 1. Hay que decir aquí las direcciones físicas de los dispositivos (de disco, de cinta, lectora, impresora, etc.), así como el nombre de los ficheros. Este nombre está grabado magnéticamente en el fichero que se pide (si es una cinta magnética o un disco), y el *supervisor* comprueba que el nombre que le ha proporcionado el operador coincide con el que hay grabado magnéticamente en el dispositivo. Si no coinciden, avisará al operador para que monte otro dispositivo. Esto evita que un programa procese, por error de montaje, ficheros magnéticos que pertenecen a otros programas.

3. El *supervisor* carga el programa 1, anota en una tabla en memoria qué dispositivos de E/S se le asignan y transfiere control a su primera instrucción.

4. Si el operador quiere ahora cargar el programa 2 a la vez que se ejecuta el programa 1, primero debe averiguar en qué sitio de memoria puede cargarlo. Para ello, pide al *supervisor* datos sobre qué posiciones de memoria no están ocupadas por el *supervisor* ni el programa 1. A la vista de esta información, el operador comprueba si hay una zona libre de memoria suficiente para albergar el programa 2. Si es así, da la orden de cargarlo, proporcionando al *supervisor* los datos que se citaban en el punto 2.

5. El *supervisor,* al cargar el programa 2, comprueba que cabe en la memoria disponible; si así no fuera dará mensaje de error al operador. Comprueba también que los dispositivos de E/S que necesita no están asignados al programa 1, pues si lo estuvieran también daría mensaje de error al operador.

En resumen: comprueba que los recursos que el programa 2 necesita están disponibles. Entonces, una vez cargado el programa 2, le transfiere control a la primera instrucción.

6. A partir de este momento, ambos programas se ejecutan a la vez, aprovechándose las esperas de uno para ejecutar instrucciones del otro, como ya hemos dicho.

7. Supongamos que acaba uno de los programas, por ejemplo, el 2 (el 1 aún sigue ejecutándose). El *supervisor* avisa al operador que el programa 2 ha terminado y que ya puede desmontar los ficheros que le pertenecían.

8. A partir de aquí los recursos que empleaba el programa 2 (memoria y dispositivos de E/S) están ya disponibles para ser utilizados de nuevo. El operador puede decidir cargar otro nuevo programa.

IDEAS BASICAS SOBRE UN SUPERVISOR PARA MULTIPROGRAMACION

Hemos dicho que cuando dos o más programas del usuario coexisten en memoria y trabajan en multiprogramación, la unidad de control va «saltando» de uno a otro, aprovechando los tiempos de espera de cada uno. ¿Quién se encarga de decidir estos «saltos»?: El *supervisor*. Esta es una de las funciones fundamentales del *supervisor* en un sistema de multiprogramación. Vamos a profundizar algo más sobre cómo el *supervisor* puede realizar esta función. El mecanismo fundamental que utiliza para ello es el de las *interrupciones*, ya explicadas.

El *supervisor* tiene una tabla en memoria de todos los programas que están cargados en un momento dado, con los datos siguientes:

Nombre del Programa	Origen en memoria	N.° posiciones de memoria que ocupa	Estados del Programa	(Otros campos)
PRO 1	4000	2000	A	—
PRO 2	6000	2000	D	—

Fig. 26. Falta en memoria de programas.

En el campo de «estado del programa» de la tabla de programas del *supervisor* se almacena un código que indica la situación de cada programa. Por ejemplo, los códigos y sus significados podrían ser:

A. *Programa activo.* Indica que la unidad de control está en este momento ejecutando instrucciones de este programa.

E. *Programa en espera.* Este programa está esperando a que ocurra algún suceso y no puede ejecutarse hasta que ocurra (por ejemplo, que acabe una operación de lectura).

D. *Programa dispuesto para ejecutarse.* El programa está listo para continuar su ejecución; no necesita esperar a que ocurra ningún suceso, pero no está activo, es decir, la unidad de control no lo está ejecutando (porque en este momento estará atareada en la ejecución de otro programa).

En la figura 26 tenemos que, según indica la tabla de programas mostrada, en este momento hay dos programas cargados en memoria: el PRO 1 y el PRO 2. El PRO 1 está cargado en memoria, ocupando las posiciones 4000 a 5999, y el PRO 2 está en las posiciones 6000 a 7999. La unidad de control está en este momento ejecutando el programa PRO 1 (programa

activo). El programa PRO 2 está listo para ser ejecutado (programa dispuesto), pero tiene que esperar a que el PRO 1 tenga un tiempo de espera para que la unidad de control lo pueda ejecutar.

Cuando un programa que esté activo detecta que necesita esperar algún suceso antes de seguir, lo comunica al *supervisor* mediante una interrupción. Por ejemplo, supongamos que PRO 1 da un READ para leer una ficha, y que necesita que se termine de leer esta ficha antes de poder continuar su proceso (por ejemplo, necesita hacer ciertos cálculos con los datos de la ficha antes de seguir). Entonces dará una macro WAIT, que mediante una instrucción SVC provocará una interrupción y, por consiguiente, se transfiere control a la rutina de control del *supervisor*. Esta rutina anotará en la tabla de programas que PRO 1 necesita esperar un suceso (programa en espera). La tabla quedará así (no se muestra más que los campos de Programa y Estado):

Programa	**Estado**
PRO 1	E
PRO 2	D

A continuación explora la tabla de programas para ver si hay alguno que está disponible para ser ejecutado (programa dispuesto). En nuestro ejemplo, PRO 2 lo está. Entonces anota en la tabla que ahora está ya activo y le transfiere control. La tabla quedará reflejando esta situación:

Programa	**Estado**
PRO 1	E
PRO 2	A

Supongamos que ahora se acaba de leer la ficha que PRO 1 estaba esperando. El fin de la operación de lectura provoca una interrupción de E/S, que transfiere el control al *supervisor*. Este trata la interrupción y modifica la tabla de programas quitando a PRO 1 del estado de espera, puesto que ya ha ocurrido el suceso que estaba esperando.

Programa	**Estado**
PRO 1	D
PRO 2	A

Como en este momento la unidad de control está ejecutando instrucciones del *supervisor*, el programa PRO 2 no está activo, sino disponible. Es decir, el *supervisor* modifica el estado de PRO 2 para que la tabla, como siempre, refleje la situación del momento:

Programa	**Estado**
PRO 1	D
PRO 2	D

Ahora el *supervisor* tiene que decidir a qué programa va a transferir control. Para ello explora la tabla y elige el programa que está disponible (estado = D). En nuestro caso, los dos están dispuestos. ¿Cuál eligirá el *supervisor*? Elige el que nos interesa. Para ello, el operador, cuando pide que se cargue un programa, le asigna una prioridad de ejecución. Esta prioridad de ejecución se almacena en la tabla de programas cuando se hace la carga. Supongamos, en nuestro caso, lo siguiente:

Programa	**Estado**	**Prioridad**
PRO 1	D	2
PRO 2		
D		
1		

El *supervisor,* entonces, elige el programa que, estando dispuesto, tiene asignada mayor prioridad (en nuestro ejemplo PRO 1) y le transfiere control, después de modificar la tabla para que refleje que PRO 1 está activo:

Programa	**Estado**	**Prioridad**
PRO 1	A	2
PRO 2	D	1

En resumen:

— Cuando un programa necesita esperar que ocurra un suceso, lo comunica al *supervisor* (mediante una interrupción).

— Cuando el suceso que está esperando un programa ocurre, se provoca una interrupción también, transfiriendo control al *supervisor.*

— El *supervisor* lleva un control continuo de qué programas hay cargados en memoria y su situación en cada momento.

— El *supervisor* es quien decide, cada vez que ocurre una interrupción, cuál programa, de los que hay cargados en memoria, se ejecutará.



REGLA DE PRIORIDADES

El objetivo de la multiprogramación es maximizar la utilización de la unidad de control.

En los párrafos anteriores hemos visto que cuando varios programas están listos para ejecutarse, el *supervisor* elige el de más prioridad. Está claro, por tanto, que este parámetro (la prioridad) influye en el grado de utilización de la unidad de control. Por otra parte, este parámetro lo fija el operador cuando se carga el programa. La pregunta que aquí nos planteamos es: ¿cómo decidir si dar más o menos prioridad a un programa, para maximizar la utilización de la unidad de control?

Supongamos que hay dos programas en memoria, PRO 1 y PRO 2, y que PRO 1 tiene más prioridad. Siempre que PRO 1 esté activo, continuará ejecutándose, aunque haya interrupciones, hasta que necesite esperar a que ocurra algún suceso. En cambio, cuando PRO 2 está activo, puede dejar de estarlo aunque no necesite esperar un suceso. En efecto, si PRO 2 está activo en un cierto momento, y ocurre una interrupción que estaba esperando PRO 1, como éste tiene más prioridad recibe control, y PRO 2 lo pierde.

Es decir, PRO 1 está activo todo el tiempo que no está esperando ningún suceso, y PRO 2 está activo cuando no espera ningún suceso, y además, PRO 1 le deja. De modo que PRO 1 se ejecuta al mismo ritmo que si estuviera solo (en monoprogramación), y PRO 2 se ejecuta aprovechando los «huecos» de PRO 1.

Lógicamente, habrá que *dar más prioridad al programa que tenga más «huecos»* (o tiempos muertos, o esperas obligadas), con objeto de que el otro tenga más oportunidades de ejecutarse.

Pondremos un ejemplo para aclarar esta regla.

Supongamos que tenemos dos programas, A y B.

El programa A lee unos números de una ficha, efectúa cálculos con ellos que duran una hora e imprime el resultado. El programa B lee 60000 fichas, cada una conteniendo un número, suma todos estos números e imprime el resultado. La velocidad de la lectora es de 1000 fichas por minuto. ¿Cúal de los programas, ejecutándose en multiprogramación, deberá tener más prioridad? Evidentemente, el B tendrá muchas esperas por operaciones de lectura, mientras que el A no tiene ninguna. Por tanto, deberíamos dar más prioridad al B. Veamos algunos diagramas de tiempos.

Si el programa A tuviera prioridad sobre el B, tendríamos (suponemos que cada programa tiene su propia lectora e impresora):

En la figura 27 se ve que B tiene que esperar durante una hora (todo el proceso de A) antes de poder continuar; si el programa B tiene más prioridad tendremos (ver figura 28):

Cuando se acaba de leer una ficha del programa B (por ejemplo, la segunda), éste recibe control y efectúa el cálculo P2. Cuando lo termina, tiene que esperar a que se acabe de leer la ficha siguiente (la tercera). Este tiempo de espera se aprovecha para seguir la ejecución de A (proceso P3). Es decir, el programa B ejecuta en el mismo tiempo que si estuviera en monoprogramación. El A algo más lento que en monoprogramación, pero no mucho, pues los intervalos de uso de unidad de control que le «roba» el B son pequeños.

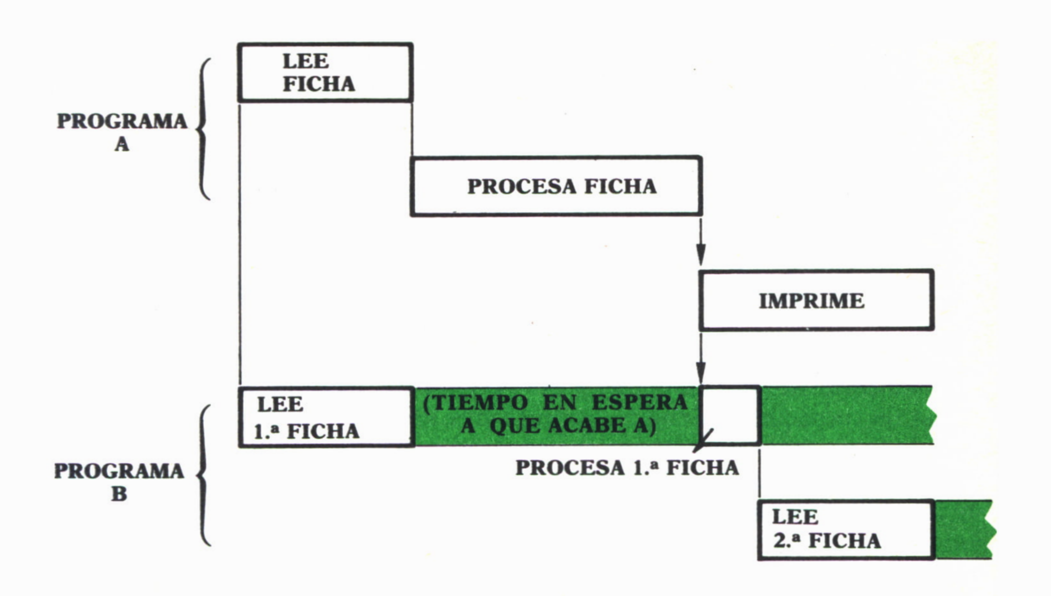

PROGRAMA A
LEE FICHA
PROCESA FICHA
IMPRIME
PROGRAMA B
LEE 1.ª FICHA
(TIEMPO EN ESPERA A QUE ACABE A)
PROCESA 1.ª FICHA
LEE 2.ª FICHA

Fig. 27.

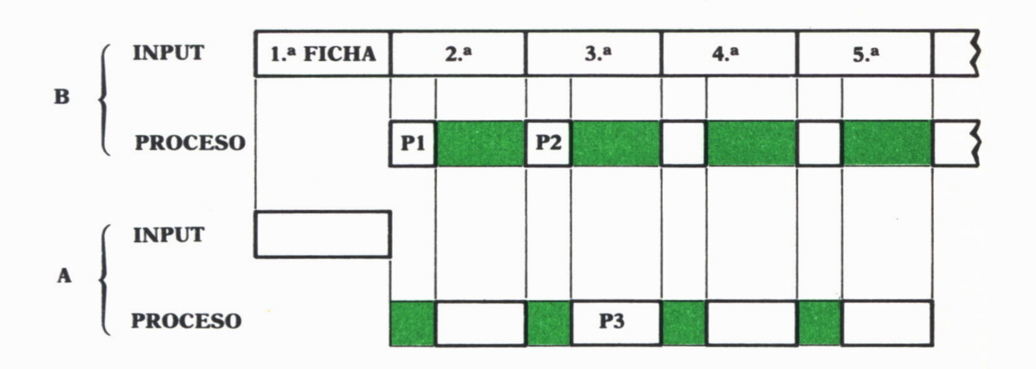

B
INPUT
1.ª FICHA
2.ª
3.ª
4.ª
5.ª
PROCESO
P1
P2
A
INPUT
PROCESO
P3

Fig. 28.



RESUMEN DEL CAPITULO

— Los programas que tienen mucha E/S utilizan muy poco la unidad de control, aunque solapen las operaciones de E/S entre sí y con el proceso.

— La multiprogramación permite mejorar el grado de utilización de la unidad de control y, por tanto, obtener un mejor rendimiento del ordenador.

— La multiprogramación consiste en tener varios programas cargados en memoria simultáneamente, y aprovechar los tiempos de espera inevitables de unos para ejecutar instrucciones de otros (la unidad de control no puede ejecutar más de una instrucción a la vez).

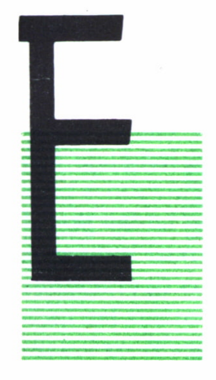
— Para realizar la multiprogramación se utiliza el mecanismo de interrupciones del hardware, y además, es necesario un software preparado para ello.

— El *supervisor* es el responsable de la gestión del control entre los programas en multiprogramación, siendo ésta una de sus funciones más importantes.

— Cuando varios programas se ejecutan en multiprogramación, hay que asignar más prioridad a los que usan menos la unidad de control.

FUNCIONES ADICIONALES DE LOS SISTEMAS OPERATIVOS





EN los capítulos anteriores hemos ido desarrollando los conceptos básicos del funcionamiento de un *supervisor*. Vamos ahora a dar una descripción breve de algunas funciones adicionales de los sistemas operativos, que no hemos explicado hasta ahora para no distraer la atención de los puntos más importantes.



SENTENCIAS DE CONTROL (JOB CONTROL LANGUAGE).

En la práctica, la misión del operador puede ser muy complicada. Varios programas se pueden estar ejecutando en un momento dado en multiprogramación, cada uno con sus ficheros propios; pueden ocurrir incidencias: un fichero contiene datos erróneos, un programa ocupa más memoria de la disponible, etc.

Todas estas situaciones imprevistas provocan mensajes al operador, que debe analizarlos y actuar en la forma adecuada.

Un objetivo fundamental del sistema operativo es tomar a su cargo el máximo posible de trabajo del operador, facilitándole a éste su labor. Esto es en gran parte posible porque muchas tareas de operación pueden programarse (normalmente se incluyen en el *supervisor*). Por ejemplo, un usuario podría dar al operador las instrucciones siguientes:

«Mi programa se llama PROG. Está escrito en FORTRAN y el programa fuente está en fichas perforadas. El compilador de Fortran está en una librería llamada FORLIB, en un disco identificado con el numero 111111. Por favor, compile el programa, y si va bien la compilación, guarde el módulo objeto en una librería llamada OBJLIB en el disco 111111. El programa compilado no excederá de 4000 posiciones de memoria. Los datos de

entrada están en la cinta magnética número 123456, y el programa necesita un carrete de cinta en blanco donde grabará información para uso posterior en otros programas.

Mi programa tiene muchos cálculos, de modo que debería ejecutarse con una prioridad baja. Si el compilador encuentra errores de sintaxis en mi programa fuente, no ejecutar el programa objeto. Mi programa produce un listado sobre el papel preimpreso, con el formato número 00018.»

Algunas de estas instrucciones hay que llevarlas a cabo a mano (montar cintas, montar papel, etc.), pero la mayoría podrían ejecutarse por un programa. Por ejemplo, podríamos incluir en el *supervisor* una rutina cuya lógica respondiera al organigrama de la figura 29.

El organigrama anterior puede ser ejecutado por el *supervisor* si éste conoce las especificaciones de ejecución del programa (por ejemplo, los discos necesarios, las cintas, etc.). Si los requerimientos de cada programa los especifica el usuario detalladamente y mediante un lenguaje formal (como si fuera un lenguaje de programación), estas instrucciones pueden plasmarse en un soporte legible por el ordenador y ser leídas por el *supervisor*, que así estará en condiciones de ejecutar las tareas indicadas en el organigrama anterior. Esto relevaría al operador de las operaciones rutinarias (cargar programas) y se podría concentrar en analizar las incidencias anormales. El lenguaje formal que se utiliza para especificar los requerimientos de ejecución de un programa se llama **lenguaje de descripción o de control de trabajos** (JOB DESCRIPTION LANGUAGE, o también a veces JOB CONTROL LANGUAGE).

El lenguaje de control de trabajos lo escribe el programador. El operador se limita a introducir las especificaciones en la lectora de fichas (si están en ficha), aunque en otras ocasiones es el propio programador quien introduce las especificaciones por teclado.



FUNCIONES BASICAS DE UN SISTEMA OPERATIVO

La función básica de un sistema operativo es controlar los recursos del ordenador para optimizar y facilitar su utilización por parte de los operadores y programadores.

Los recursos a controlar y manejar son:

— Memoria.
— Uso de la unidad central.
— Dispositivos de E/S.
— Canales de E/S.

Las diversas rutinas y programas que componen el sistema operativo

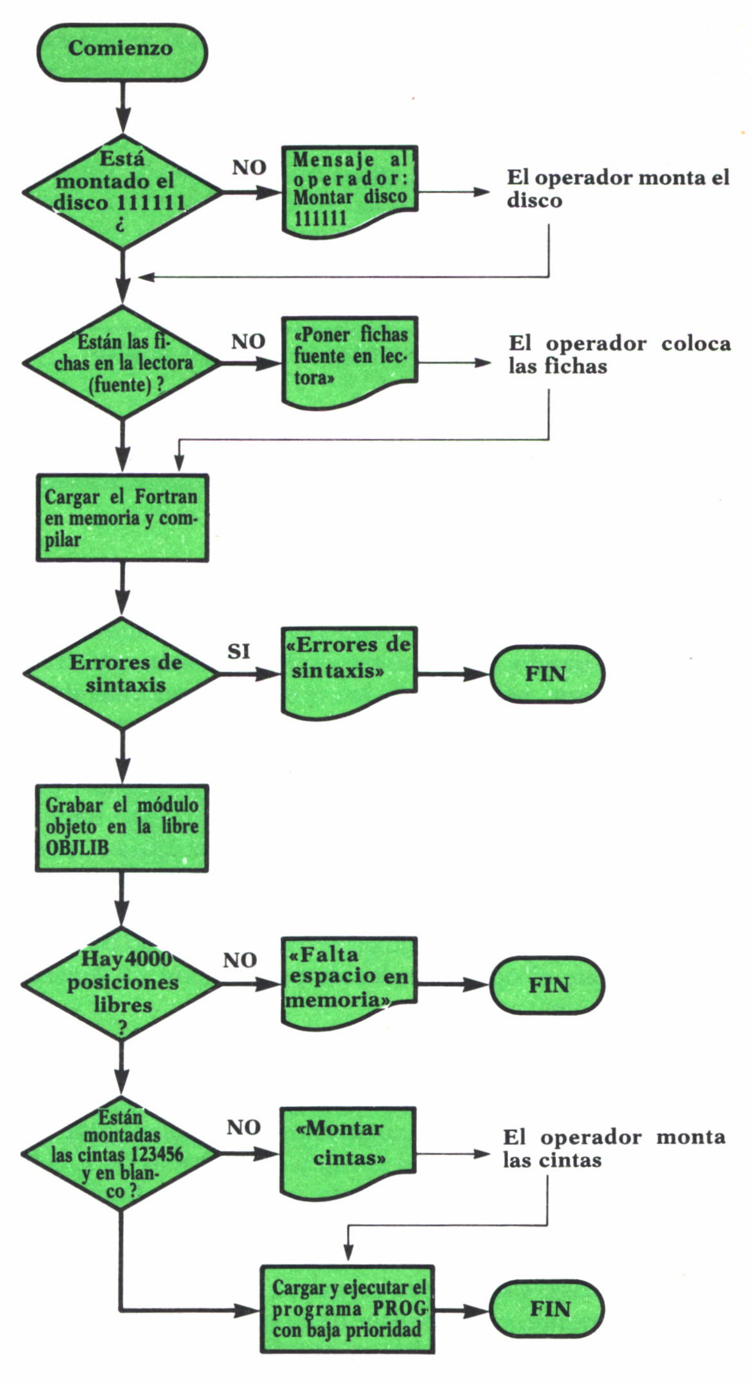

Comienzo
Está montado el disco 111111 ¿
NO
Mensaje al operador: Montar disco 111111
El operador monta el disco
Están las fichas en la lectora (fuente) ?
NO
«Poner fichas fuente en lectora»
El operador coloca las fichas
Cargar el Fortran en memoria y compilar
Errores de sintaxis
SI
«Errores de sintaxis»
FIN
Grabar el módulo objeto en la libre OBJLIB
Hay 4000 posiciones libres ?
NO
«Falta espacio en memoria»
FIN
Están montadas las cintas 123456 y en blanco ?
NO
«Montar cintas»
El operador monta las cintas
Cargar y ejecutar el programa PROG con baja prioridad
FIN

Fig. 29.

cumplen unas funciones de control de los recursos anteriores, que pueden agruparse en las siguientes categorías:

— Gestión de trabajos.
— Gestión de memoria.
— Gestión de multiprogramación.
— Gestión de dispositivos de E/S.
— Gestión de canales y operaciones de E/S.
— Gestión de datos.
— Gestión de la comunicación con el operador.

Vamos a comentar brevemente alguno de estos puntos.



GESTION DE TRABAJOS

Hemos visto que cada trabajo a realizar se entrega al operador en forma de especificaciones en un lenguaje especial para este fin (JOB CONTROL LANGUAGE). Supongamos que estas especificaciones están perforadas en fichas. Cuando el operador decide ejecutar este trabajo, da un mensaje al *supervisor,* indicándolo así, y coloca las fichas de las especificaciones en la lectora. Entonces el *supervisor* transfiere control a una rutina suya, que se suele llamar *reader.* Esta rutina lee las fichas con las especificaciones del trabajo (donde se indica el nombre del programa, su prioridad, etc.). A partir de aquí el sistema operativo se hace cargo de la ejecución del trabajo. Mientras se ejecuta este trabajo, el operador puede poner en la lectora las especificaciones de otros trabajos, que serán leídas por el *reader* y pasadas al *supervisor,* para que éste inicie su ejecución en multiprogramación. El *reader* seguirá activo, preparado para leer fichas de la lectora, hasta que el operador decida pararlo con un mensaje desde el teclado de la consola.

En resumen, el operador introduce en la lectora de una vez todas las especificaciones de trabajos que tenga, y arranca el *reader,* guarda todas las fichas leídas en un disco (en un fichero que suele llamarse *cola de entrada de trabajos*). El *supervisor* carga en memoria y ejecuta el primer trabajo de la cola. Mientras haya memoria y dispositivos de E/S disponibles, carga en multiprogramación el segundo, tercero, etc. Cuando no haya memoria o dispositivos de E/S tiene que esperar a que acabe alguno de los que se están ejecutando y se liberen los recursos que tenía asignados (memoria y dispositivos), y entonces carga en memoria el siguiente trabajo, etcétera. Naturalmente, el *supervisor* tiene que comunicar al operador qué trabajos arranca en cada momento y qué ficheros hay que montar y desmontar.



GESTION DE MEMORIA

Cuando la memoria está llena con varios programas ejecutándose, hay que esperar a que alguno de ellos acabe antes de cargar el siguiente de la cola de entrada, como ya hemos dicho. No obstante, puede ocurrir que no quepa en el hueco producido. Por ejemplo, supongamos cinco programas ejecutándose en multiprogramación (P1, P2, P3, P4, P5).

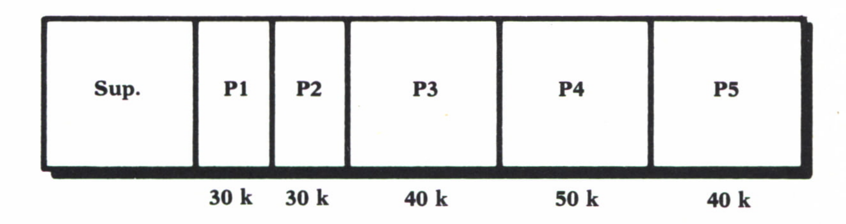


Fig. 30. Programas ejecutándose en multiprogramación.

Supongamos que el siguiente programa a ejecutar, P6, necesita 100K y que acaba P1. Naturalmente, P6 no puede cargarse en el hueco dejado por P1, y debe esperar a que se produzcan más huecos. Supongamos que acaban P3 y P5. Tendremos:

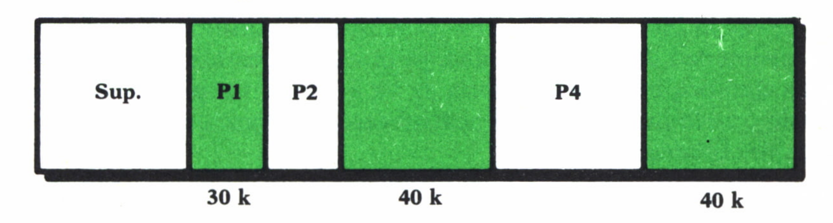


Fig. 31. P_3 y P_5 finalizan.

Se ve en la figura anterior que tendríamos disponibles 30+40+40 = 110k, suficientes para cargar P6. Pero no puede cargarse porque no son contiguas las áreas disponibles. El programa P6 se ve obligado a esperar. a esperar.

Este problema, conocido como *fragmentación* de la memoria, se puede resolver con uno de los siguientes métodos:

Por compactación

Consiste en hacer que el *supervisor traslade los programas P2 y P4 hacia las zonas altas de memoria y los junte.*

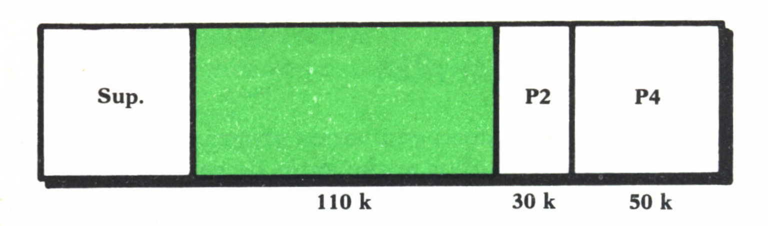


Fig. 32. Método de compactación.

Por segmentación del programa

Consiste en hacer posible que el programa se ejecute en zonas no contiguas de memoria. En nuestro ejemplo, el programa se cargaría en tres zonas separadas. Normalmente, esto se lleva a efecto mediante *técnicas de segmentacion y paginación,* que requieren un registro especial de hardware para reubicar las direcciones de los programas. Este enfoque lleva el concepto, más amplio, de *memoria virtual.*



GESTION DE MULTIPROGRAMACION

Para funcionar en multiprogramación, los programas deben ser reubicables, es decir, al cargarlos en memoria del *loader* deben ser reajustadas sus direcciones de acuerdo con la posición origen de memoria donde se cargan. Esto ya fue explicado en el capítulo 2. La multiprogramación exige esta propiedad de reubicabilidad, ya que es imposible en la práctica predecir en qué posición de memoria se va a cargar un programa (depende de donde quede hueco libre al acabar otro programa anterior, lo que es prácticamente aleatorio).

Si un programa tiene un bucle, por error en su lógica, por ejemplo, una instrucción de saltar a sí mismo, su ejecución no termina nunca. Evidentemente, ésta es una situación errónea que hay que detectar de algún modo. Para ello, el *supervisor* suele llevar un control del tiempo de unidad de control empleado en la ejecución del programa desde que éste se carga. Mediante un parámetro en las especificaciones del trabajo puede especificarse un límite para este tiempo, de modo que si supera (por ejemplo, por encerrarse en la ejecución indefinida de un bucle), el *supervisor* cancela el trabajo.



GESTION DE DISPOSITIVOS DE E/S

El *supervisor* lleva un control en todo momento de los dispositivos de E/S asignados a cada programa en ejecución, como ya hemos dicho.

Además, lleva un control del espacio disponible en los discos montados, de tal modo que si se necesita espacio para un nuevo fichero, él puede buscarlo y asignarlo, evitando al operador llevar una cuenta detallada.

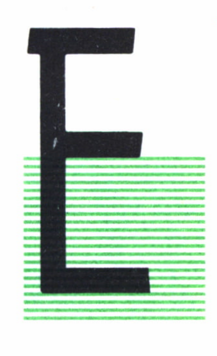


GESTION DE DATOS

Los datos almacenados en ficheros pueden estar organizados según determinados criterios, que se suelen denominar *métodos de organización y acceso* de ficheros.

En la organización secuencial indexada suele ser responsabilidad del *supervisor* el manejo de los índices, de forma que el programa se limita a dar un *READ* pidiendo el registro con una clave determinada. El *supervisor* recibe esta petición de lectura, y se encarga de consultar los índices, averiguar en qué sector del disco se encuentra la información solicitada y leerla.

INTRODUCCION AL DOS 7



EL DOS está formado por dos conjuntos de programas. El primero, el sistema de entrada/salida; el segundo, un conjunto de programas auxiliares, también llamados comandos.

SISTEMA DE ENTRADA/SALIDA

El sistema de E/S es un conjunto de programas que se leen automáticamente en la memoria del ordenador desde un disco del sistema cada vez que se arranca el ordenador.

Este sistema de E/S sirve como «interface» entre los programas del usuario y otros programas, y el «hardware». Permite usar el ordenador de la manera más adecuada.



PROGRAMAS AUXILIARES

El sistema de E/S incluye un fichero intérprete de comandos llamado COMMAND.COM, que permite al usuario llamar a cualquiera de los programas auxiliares con sólo teclear un comando desde el teclado.

Estos programas auxiliares o comandos se clasifican en dos grupos:

— Internos.
— Externos.

Los primeros se ejecutan inmediatamente porque están incluidos dentro del DOS. Por ello, una vez cargado el DOS no se necesita que el «diskette» del DOS permanezca insertado en la unidad para ejecutar estos comandos.

Los segundos residen en el «diskette» como archivos de programas; por tanto, deben leerse del «diskette» antes de ejecutarse. Esto significa que el «diskette» del DOS debe estar en la unidad, pues de no ser así el DOS no encontrará el comando.

Cualquier archivo con extensión .BAT, .COM, .EXE se considera un comando externo.

Para llamar a un comando el usuario debe teclear el nombre del mismo. Este nombre puede ir seguido de uno o más parámetros, necesarios para indicar al comando dónde debe coger los datos, dónde debe ponerlos, etc.



PROCESO POR LOTES

A menudo se tendrá que usar la misma secuencia de comandos para realizar una tarea rutinaria. El DOS incluye una prestación que simplifica las tareas repetitivas. Esta prestación se denomina *proceso por lotes.*

El proceso por lotes permite ejecutar automáticamente un grupo de comandos del DOS. En vez de teclear uno por uno los nombre de los comandos cada vez que se necesiten, la secuencia de nombres de comandos se pone en un fichero especial llamado *fichero de proceso por lotes.* Al teclear el nombre del fichero por lotes los comandos de dicho fichero se ejecutan como si se hubieran tecleado los nombres de los comandos directamente desde el teclado.

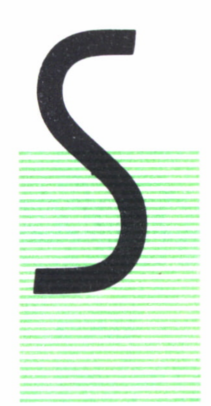


CARACTERISTICAS ESPECIALES DE CONFIGURACION

Normalmente, al arrancar un sistema operativo se deben cargar determinados ficheros y definir los parámetros necesarios para configurar el sistema, de acuerdo con la aplicación específica en uso.

El DOS incluye un conjunto de ficheros de comandos que configuran automáticamente el sistema nada más arrancar. El sistema de configuración permite al usuario empezar a trabajar en su aplicación sin tener que instalar cada aplicación específica.

COMANDOS DEL DOS 8



E presenta a continuación una descripción simplificada de los comandos más importantes del DOS. Los comandos aparecen en orden alfabético.

ASSIGN

Funcion: Indica al DOS que transforme las peticiones de entrada/salida para una unidad, en peticiones de entrada/salida para otra unidad.
Tipo: Externo.
Ejemplos:

A > Assign A=B
— El DOS envía a la unidad B las peticiones de la unidad A; así si se ejecuta DIR A, el DOS visualizará el directorio que está en la unidad B.

A > Assign
— Anula toda asignación de unidades hecha con anterioridad.

ATTRIB

Función: Permite definir el atributo de un archivo como de sólo lectura o visualizar el atributo de un archivo.
Tipo: Externo.
Ejemplos:

A > ATTRIB +R ARCHI1.TXT
— Define el archivo ARCHI1.TXT como de sólo lectura.

A > ATTRIB ARCHI1.TXT
— Lista el atributo del archivo especificado.

A > ATTRIB - R ARCHI1.TXT
— Suprime el atributo de sólo lectura del archivo especificado.

BACKUP

Función: Realiza copias de seguridad de uno o más archivos de un disco a otro disco.
Tipo: Externo.
Ejemplos:

A > BACKUP B: A: / S
— Hace una copia de seguridad de todos los archivos (S), que se encuentran en el «diskette» de la unidad B, en el «diskette» de la unidad A.

A > BACKUP A: ARCHI1.TXT B:
— Hace una copia de seguridad del archivo especificado, ubicado en la unidad A, en la unidad B.

BREAK

Función: Permite indicar ál DOS que compruebe si se ha efectuado una ruptura de control, siempre que un programa pida al DOS que realice una función
Tipo: Interno.
Ejemplos:

A > BREAK = ON
— El DOS comprobará si se está introduciendo un Ctrl-Break desde el teclado.

A > BREAK = OFF
— El DOS comprobará rupturas de control para:

- Operaciones estándar de entrada/salida.
- Operaciones estándar a dispositivos de impresión.
- Operaciones estándar a dispositivos auxiliares.

A > BREAK
— Visualiza el estado actual (ON- OFF) del comando BREAK.

CHKDSK

Función: Analiza los directorios, los archivos y la tabla de asignación de archivos en la unidad designada o asumida por omisión, produciendo un informe del estado de la memoria y del «diskette».
Tipo: Interno.
Ejemplos:

A > CHKDSK B:
— Produce un informe de estado para un «diskette» en la unidad B

A > CHKDSK B: / F
— Produce un informe de estado para un «diskette» en la unidad B,

analizando los errores encontrados en el directorio de la tabla de asignación de archivos.

A > CHKDSK B: / V
— Produce un informe de estado para el «diskette» de la unidad **B** y visualiza todos los archivos de dicha unidad.

A > CHKDSK B: *.*
— Produce un informe de estado para el «diskette» de la unidad **B** y lista los nombres de archivos que ocupan áreas no contiguas.

CLS

Función: Borrar la pantalla.
Tipo: Interno.
Ejemplo:

A > CLS
— Borra la pantalla.

COMP

Función: Compara el contenido del primer conjunto de archivos especificados con el contenido del segundo conjunto de archivos especificados.
Tipo: Externo.
Ejemplos:

A > COMP B : *.ASM A:
— Compara todos los archivos con extensión ASM del «diskette» de la unidad B con los archivos del mismo nombre del «diskette» ubicado en la unidad A.

A > COMP B : *.ASM A: *.BAK
— Compara todos los archivos ASM del «diskette» ubicado en la unidad B con todos los archivos del mismo nombre y extensión BAK del «diskette» ubicado en la unidad A.

A > COMP B : ARCHI.TXT A: ARCHI1.ASM
— Compara el archivo ARCHI.TXT del «diskette» de la unidad B con el archivo ARCHI1.ASM del «diskette» de la unidad A.

COPY

Función: Copia uno o más archivos a un «diskette» específico.
Tipo: Interno.
Ejemplos:

A > COPY B: MIPROG

— Copia el archivo MIPROG desde el «diskette» ubicado en la unidad B al «diskette» ubicado en la unidad A, con el mismo nombre.

A > COPY *.* B:

— Copia todos los archivos desde el «diskette» ubicado en la unidad A al «diskette» ubicado en la unidad B, con los mismos nombres y extensiones.

A > COPY MIPROG.ABC B : *.XXX

— Copia el archivo MIPROG.ABC desde el «diskette» de la unidad A al «diskette» de la unidad B, llamando a la copia MIPROG.XXX

A > COPY MIPROG.ABC B : ARCHI.ASM

— Copia el archivo MIPROG.ABC desde el «diskette» de la unidad A al «diskette» de la unidad B, llamando a la copia ARCHI.ASM

A > COPY ARCHI1.XYZ + ARCHI2.ABC + ARCHI3.JKL GRANARCH.TXT

— Crea un nuevo archivo llamado GRANARCH.TXT en el «diskette» de la unidad A, en el cual están los tres archivos especificados con anterioridad concatenados.

A > COPY K.ASM + J.ASM

— Añade el archivo J.ASM al final del archivo K.ASM, y deja el resultado en K.ASM

CTTY

Función: Cambia la consola de entrada/salida estándar a una consola auxiliar o restaura el teclado y la pantalla como dispositivos de entrada/salida estándar.

Tipo: Interno.

Ejemplos:

A > CTTY AUX

— El DOS utilizará el dispositivo AUX para sus operaciones de entrada/salida.

A > CTTY CON

— Invierte la asignación anterior haciendo que el DOS vuelva a utilizar para sus operaciones de entrada/salida la pantalla y el teclado.

DATE

Función: Permite introducir la fecha o cambiar la que tiene el sistema.

Tipo: Interno.

Ejemplo:

A > DATE

— Aparecen los siguientes mensajes:

«La fecha actual es: DD/MM/AA»
«Introduzca nueva fecha: DD/MM/AA»:

DEL

Función: Suprime un archivo especificado.
Tipo: Interno.
Ejemplos:

A > DEL B : ARCH.BAT
— Suprime el archivo ARCH.BAT del «diskette» ubicado en la unidad B.

A > DEL *.BAT
— Suprime todos los archivos con extensión .BAT del «diskette» ubicado en la unidad A.

A > DEL
— Suprime todos los archivos del «diskette» de la unidad A.

A > DEL *.*
— Suprime todos los archivos del «diskette» de la unidad A.

A > DEL B : ARCHI.*
— Suprime todos los archivos con nombre ARCHI del «diskette» de la unidad B, sea cual sea su extensión.

DIR

Función: Hace un listado de todas las entradas del directorio o sólo de aquéllas que corresponden a archivos especificados.
Tipo: Interno.
Ejemplos:

A > DIR
— Lista todos los archivos del «diskette» de la unidad A.

A > DIR B :
— Lista todos los archivos del «diskette» ubicado en la unidad B.

A > DIR *.ATP
— Lista todos los archivos del «diskette» de la unidad A con extensión ATP

A > DIR ARCHI.*
— Lista todos los archivos del «diskette» ubicado en la unidad A con nombre ARCHI, sea cual sea su extensión.

DISKCOMP

Función: Compara dos «diskettes».
Tipo: Externo.
Ejemplos:

A > DISKCOMP A : B :
— Compara un «diskette» ubicado en la unidad A con otro ubicado en la unidad B.

A > DISKCOMP
— Compara un «diskette» ubicado en la unidad A con otro «diskette» que deberá ser insertado en la misma unidad.

A > DISKCOMP B :
— Compara un primer «diskette» insertado en la unidad B, con un segundo «diskette» ubicado en la unidad A.

DISKCOPY

Función: Copia el contenido del «diskette» de la unidad origen en el «diskette» de la unidad destino. Si es necesario, el «diskette» destino es formateado durante la copia.
Tipo: Externo.
Ejemplos:

A > DISKCOPY A : B :
— Copia el contenido de un «diskette» ubicado en la unidad A (origen) a otro «diskette» ubicado en la unidad B (destino).

A > DISKCOPY
— Copia el contenido de un «diskette» ubicado en la unidad A, a otro «diskette» a insertar en la misma unidad.

ERASE

Función: Borra un archivo especificado.
Tipo: Interno.
Ejemplos:

A > ERASE A : ARCHI.BAT
— Borra el archivo ARCHI.BAT del «diskette» ubicado en la unidad A.

A > ERASE B : ARCHI.BAT
— Borra el archivo ARCHI.BAT del «diskette» ubicado en la unidad B.

A > ERASE *.BAT
— Borra todos los archivos con extensión .BAT del «diskette» ubicado en la unidad A.

A > ERASE ARCHI.*

— Borra todos los archivos de nombre ARCHI, sea cual sea su extensión, del «diskette» ubicado en la unidad A.

A > ERASE *.*
— Borra todos los archivos del «diskette» ubicado en la unidad A.

A > ERASE
— Realiza la misma función del ejemplo anterior.

FORMAT

Función: Inicializa el «diskette» de la unidad especificada con un formato de registro aceptable para el DOS.
Tipo: Externo.
Ejemplos:

A > FORMAT
— Formatea el «diskette» ubicado en la unidad A.

A > FORMAT B :
— Formatea el «diskette» ubicado en la unidad B.

A > FORMAT B :/1
— Formatea el «diskette», de una cara, ubicado en la unidad B al margen del tipo de unidad.

A > FORMAT B :/8
— Formatea el «diskette» ubicado en la unidad B con el formato de 8 sectores por pista.

A > FORMAT B :/V
— Formatea el «diskette» ubicado en la unidad B permitiendo darle al «diskette» una etiqueta de volumen (identificadora de «diskette»).

A > FORMAT B :/B
— Formatea el «diskette» ubicado en la unidad B, con 8 sectores por pista, y asignación de espacio para los módulos del sistema.

GRAFTABL

Función: Carga en memoria una tabla de datos de caracteres adicional para el adaptador color/gráficos.
Tipo: Externo.
Ejemplo:

A > GRAFTABL
— Carga la tabla, especificada anteriormente, en memoria.

KEYBXX

Función: Carga un programa de teclado que sustituye al programa de teclado residente en ROM BIOS.

La (XX) representa uno de los cinco programas de teclado existentes:

— UK (Reino Unido)
— GR (Alemania)
— FR (Francia)
— IT (Italia)
— SP (España)

Tipo: Externo.

Ejemplos:

A > KEYBSP
— Carga el programa de teclado español.

LABEL

Función: Permite crear, cambiar y suprimir la etiqueta de volumen de un «diskette» (ésta se emplea para identificar un «diskette» mediante un nombre).
Tipo: Externo.

Ejemplos:

A > LABEL B: PROGRAMAS
— Crea etiqueta de volumen «PROGRAMAS» para el «diskette» insertado en la unidad B.

A > LABEL B:
— Borra la etiqueta de volumèn en el «diskette» de la unidad B (si la tuviese).

MORE

Función: Leer datos del dispositivo de entrada estándar, enviar una pantalla completa de datos al dispositivo de salida estándar y luego hacer una pausa con el mensaje .... MAS ....
Tipo: Externo.

Ejemplo:

A > MORE < TEST.ASM
— Se visualizará el contenido del archivo TEST.ASM con una pantalla completa cada vez. Cuando se llene la pantalla aparecerá el mensaje .... MAS ...., pulsando cualquier tecla podremos ver la siguiente pantalla completa.

PROMPT

Función: Establece un nuevo símbolo indicador del sistema.
Tipo: Externo.
Ejemplo:

A > PROMPT ABC
— ABC pasa a ser el nuevo indicador del sistema sustituyéndolo por A>.

RECOVER

Función: Recuperar archivos procedentes de un «diskette» que tenga algún sector defectuoso. Puede recuperarse el archivo que contiene el/los sector/res no válidos (menos los datos de dicho sector), o pueden recuperarse todos los archivos del «diskette» si se dañó el directorio.
Tipo: Externo.
Ejemplos:

A > RECOVER B : MIPROG
— Hace que se lea sector a sector el archivo MIPROG del «diskette» de la unidad B, saltándose los sectores defectuosos.

A > RECOVER B :
— Recupera el contenido de todo un «diskette» procedente de la unidad B.

RENAME (REM)

Función: Cambia el nombre y/o extensión de un archivo especificado.
Tipo: Interno.
Ejemplo:

A > RENAME B : TEXTO CARTA
— Cambia el nombre del archivo, existente en el «diskette» de la unidad B, con nombre TEXTO por CARTA.

RESTORE

Función: Restaura uno o más archivos de un «diskette» copia de seguridad a otro «diskette».
Tipo: Externo.
Ejemplos:

A > RESTORE A : B : *.DAT
— Restaura los archivos del «diskette» insertado en la unidad A, que tienen extensión .DAT, en el «diskette» de la unidad B.

A > RESTORE B : A : TEXTO.DAT

— Restaura el archivo TEXTO.DAT del «diskette» insertado en la unidad B, al «diskette» de la unidad A.

SHARE

Función: Carga el soporte para compartir archivos.
Tipo: Externo.
Ejemplos:

A > SHARE
— Carga el soporte de archivos compartidos.

SORT

Función: Lee datos del dispositivo de entrada estándar, los clasifica y luego los escribe en el dispositivo de salida estándar.
Tipo: Externo.
Ejemplos:

A > SORT < ARCHI1.TXT > ARCHI2.TXT
— Lee el archivo ARCHI1.TXT, hace una clasificación del mismo y luego escribe el resultado en el archivo ARCHI2.TXT.
Nota: Si en el ejemplo anterior especificamos:
A > SORT / R < ARCHI1.TXT > ARCHI2.TXT

la clasificación se hará en forma invertida.

SYS

Función: Transfiere los archivos del sistema operativo de un "diskette" a otro.
Tipo: Externo.
Ejemplo:

A > B : SYS A :
— Transfiere los archivos del sistema operativo del «diskette» de la unidad B al «diskette» de la unidad A.

TYPE

Función: Muestra el contenido de un archivo especificado en el dispositivo de salida estándar.
Tipo: Interno.
Ejemplo:

A > TYPE B : PROG1.DAT
— El contenido del archivo PROG1.DAT que está en el «diskette» de la unidad B se visualiza en el dispositivo estándar de salida.

VERIFY

Función: Verifica que los datos grabados en un “diskette“ han sido correctamente registrados.
Tipo: Interno.
Ejemplos:

A > VERIFY ON
— Activa el comando.

A > VERIFY OFF
— Desactiva el comando.

A > VERIFY
— Visualiza el estado actual del comando.

VOL

Función: Visualiza la etiqueta de volumen del «diskette».
Tipo: Interno.
Ejemplos:

A > VOL
— Visualiza la etiqueta del «diskette» de la unidad A.

A > VOL B :
— Visualiza la etiqueta del «diskette» de la unidad B.

COMANDOS DE PROCESO POR LOTES 9

FICHEROS DE PROCESO POR LOTES DEL DOS



LOS ficheros de proceso por lotes del DOS contienen una serie de comandos que se utilizan con mucha frecuencia.

Para llamar a estos comandos, basta con teclear el nombre del fichero, en vez de tener que teclear cada comando individualmente.

Cuando se crea un fichero de este tipo, se deberá incluir siempre la extensión .BAT en el nombre del fichero.

El DOS dispone de comandos especiales pensados específicamente para su uso en ficheros de proceso por lotes. Estos comandos controlan el proceso por lotes desde dentro del propio fichero y permiten modificar la secuencia de comandos del DOS del fichero de proceso por lotes, según las necesidades que existan en el momento.

También existen comandos de proceso por lotes que permiten visualizar mensajes desde dentro del propio fichero. Estos comandos informarán del desarrollo del proceso por lotes. También se usan para pedir la intervención del usuario en el proceso cuando, por ejemplo, sea necesario cambiar los «diskettes».



EL FICHERO AUTOEXEC.BAT

El fichero AUTOEXEC.BAT es un fichero de proceso por lotes que se ejecuta automáticamente al arrancar el DOS.

También se puede ejecutar este fichero en cualquier momento con sólo teclear el nombre AUTOEXEC.



COMO CREAR UN FICHERO DE PROCESO POR LOTES

Un fichero de proceso por lotes puede tener cualquier nombre, pero debe incluir la extensión .BAT . Desde el teclado, se puede crear un fichero de proceso por lotes de varias maneras:

— Usando el comando COPY.
— Usando el editor de líneas EDLIN.
— Mediante cualquier otro programa editor o de proceso de texto.

Ejemplo:

COPY CON : BATFILE.BAT
CLS
DATE
TIME
F6

Este ejemplo crea, mediante el comando COPY, el fichero de proceso por lotes BATFILE.BAT.

El fichero de proceso por lotes se termina con un carácter de fin de fichero. Este carácter se consigue pulsando F6.



COMANDOS

Todos los comandos que se describen a continuación son internos:

ECHO

Formato: ECHO [ON] [OFF] [textomensaje]
Función: Activa o desactiva la visualización en pantalla de los comandos de proceso por lotes y del DOS, según se van leyendo en el fichero.

Notas: Si se especifica el parámetro ON, todas las líneas de comandos aparecerán en pantalla a medida que se vayan leyendo en el fichero de proceso por lotes.

Especificando el parámetro OFF en pantalla, sólo aparecerán los mensajes del sistema y de error que se produzcan.

Si se especifica el parámetro «textomensaje», aparecerá en la pantalla el mensaje especificado.

FOR

Formato: FOR % % variable IN (set) DO comando
Función: Permite la ejecución repetida de un comando del DOS.

GOTO

Formato: GOTO etiqueta
Función: Transfiere el control a línea de comandos que sigue a la línea que contiene la etiqueta especificada.

IF

Formato: IF [NOT] condición comando
Función: Permite una ejecución condicional de los comandos del DOS.

Notas: Cuando la condición especificada por la instrucción IF sea verdadera, se ejecutará el comando del DOS especificado a continuación. Si la condición es falsa, se pasará por alto este comando y se ejecutará el siguiente.

PAUSE

Formato: PAUSE [textomensaje]
Función: Permite que el usuario intervenga durante la ejecución de un fichero de proceso por lotes.

Notas: Cuando el DOS encuentra este comando detiene el proceso del fichero y muestra un mensaje pidiendo que el usuario pulse cualquier tecla para continuar la ejecución.

El parámetro «texto mensaje» es opcional.

REM

Formato: REM [textomensaje]
Función: Muestra mensajes de un fichero de proceso por lotes.

Notas: Cuando el DOS encuentra la instrucción REM aparecerá en la pantalla tanto la palabra clave REM como el mensaje especificado.

SHIFT

Formato: SHIFT
Función: Permite incluir más de los diez parámetros normalmente emitidos en el fichero de proceso por lotes.

UTILIDADES 10

EDLIN



A utilidad **EDLIN** es un editor de líneas que puede usarse para crear, modificar y ver en pantalla programas fuente o ficheros de texto. Los usos específicos del **EDLIN** son:

— Crear y salvar nuevos ficheros.
— Actualizar ficheros ya existentes y salvar tanto los ficheros actualizados como los originales.
— Ver el contenido de los ficheros para borrar, insertar o sustituir texto.

El texto de estos ficheros está dividido en líneas numeradas de 253 caracteres cada una, como máximo.

Con **EDLIN**, las líneas se teclean y editan de una en una. Para editar el texto de una línea, se usarán las *teclas de función de edición y de control del DOS*.

A continuación damos información común a todos los mandatos del EDLIN:

— Excepto el mandato E (editar línea), todos los mandatos consisten en una única letra.
— Salvo los mandatos E (editar línea) y Q (abandonar editor), los mandatos van normalmente precedidos y/o seguidos por parámetros.
— Aunque sólo se requiere un separador entre dos números de línea adyacentes, conviene separar los mandatos y los parámetros con separadores para hacerlos más legibles.
— Los mandatos pasan a ser efectivos sólo después de que se pulsa la tecla <intro>.
— Los mandatos se terminan pulsando las teclas Ctrl-Break.
— Para aquellos mandatos que dan lugar a una salida mayor que la pantalla, debe pulsarse Ctrl-Numlock para detener la visualización. Pulsando cualquier otro carácter, se reanudará la visualización.
— El símbolo indicador de EDLIN es *.

— Es posible hacer referencia a números de línea relativos respecto de la línea actual. Se usará un signo (–) y un número de línea para indicar una línea que esté antes de la línea actual. Se usará un signo (+) y un número para indicar una línea que esté detrás de la línea actual.

Por ejemplo:

+ 10 visualiza las diez líneas posteriores a la línea actual.

— En una línea de mandatos pueden introducirse varios mandatos.

— Pueden insertarse en el texto caracteres de control. Para introducir un carácter de control, se pulsará CTRL-V, luego se introducirá el carácter de control deseado en mayúscula.

Mandato	**Formato**
Append Lines *Añadir Líneas*	[*n*]A
Copy Lines *Copiar Líneas*	[*línea*],[*línea*],*línea*[*veces*]C
Delete Lines *Borrar Líneas*	[*línea*][,*línea*]D
Edit Lines *Editar Líneas*	[*línea*]
End Edit *Fin Edición*	E
Insert Lines *Insertar Líneas*	[*línea*]I
List Lines *Listar Líneas*	[*línea*][,*línea*]L
Move Lines *Mover Líneas*	[*línea*],[*línea*],*línea*M
Page *Página*	[*línea*][,*línea*]P
Quit Edit *Abandonar Editor*	Q
Replace Text *Reemplazar Texto*	[*línea*][,*línea*][?]R[*serie*][< F6 >*serie*]
Search Text *Buscar Texto*	[*línea*][.*línea*][?]S[*serie*]
Transfer Lines *Transferir Líneas*	[*línea*]T *nombarch*[.*ext*]
Write Lines *Escribir Líneas*	[*n*]W



LINK

Esta utilidad combina distintos ficheros objeto en un módulo ejecutable, el cual puede cargarse en memoria y ejecutarse como si fuera un programa de aplicación.



EXE2BIN

Esta utilidad convierte ficheros con extensión .EXE de segmentos reubicables, tal como los genera la utilidad LINK, a un formato de imagen de memoria compatible con los ficheros .COM.

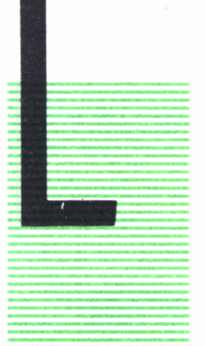


DEBUG

Esta utilidad comprueba y depura, para conseguir su funcionamiento correcto, módulos de programa de aplicación producidos con la utilidad LINK.

CONFIGURACION 11

INTRODUCCION



LOS ficheros de configuración configuran e inicializan automáticamente el DOS cada vez que se arranca el sistema.

Algunos de los ajustes que pueden configurarse son:

— La versión nacional del teclado.
— La forma en que se teclea y se visualiza la información específica de cada país.
— El número de ficheros que pueden abrirse simultáneamente.
— El número de memorias intermedias de disco asignadas en memoria.

Otros ejemplos son controladores opcionales de dispositivos, como son los controladores para reloj de tiempo real y un disco RAM.



FICHEROS DE CONFIGURACION

— PRESTART.SYS
— CONFIG.SYS

Estos dos ficheros inicializan el DOS para las necesidades específicas. Los ficheros se configuran antes de vender el equipo y en la mayoría de los casos, no hace falta modificarlos.



FICHERO PRESTART.SYS

El fichero PRESTART.SYS puede incluir los nombres de los siguientes comandos con una extensión .COM:

KEYB

Este comando ya ha sido descrito en el capítulo II (comandos del DOS) de este libro.

GRAFTABL GRAFCHAR.SYS

El comando GRAFTABL carga el fichero GRAFCHAR.SYS, que contiene un generador completo de caracteres para escritura en modos gráficos.

FONT 1

Si este comando está incluido, se usará el generador de caracteres danés/noruego. Si el comando FONT no está incluido en el fichero, se usará el juego internacional de caracteres.

Nota: Cualquier comando externo o programa cuya extensión de nombre de fichero sea .COM puede estar incluido en el fichero PRESTART.SYS.



CONFIG.SYS

El fichero CONFIG.SYS contiene un solo comando: COUNTRY =.

Sin embargo, puede incluir cualquiera de estos otros comandos:

— BREAK =
— BUFFERS =
— DEVICE =
— FILES =
— SHELL =

Los comandos del fichero CONFIG.SYS no son comandos normales, ya que no corresponden a un programa como el resto de los comandos. Simplemente contienen parámetros que el DOS interpreta.

BREAK =

Formato: BREAK = ON/OFF
Función: Especifica las condiciones para la comprobación del teclado con Ctrl-Break.

Nota: Para más información consultar capítulo 8, donde se describe el comando BREAK.

BUFFERS =

Formato: BUFFERS = nn
Función: Permite asignar (en la memoria) un número específico de me-

morias intermedias de entrada y salida para operaciones de lectura o escritura de discos.

Notas: El parámetro nn especifica el número de memorias intermedias y puede tomar cualquier valor de 1 a 99. Su valor por defecto es 2.

COUNTRY =

Formato: COUNTRY = nn

Función: Determina el modo en que se teclea y muestra en pantalla la fecha y la hora.

Notas: El parámetro nn puede tomar los siguientes valores: 1,33,34,39,44,45,46,47,49. El valor por defecto es 1.

DEVICE =

Formato: DEVICE = [u:] [ruta] nombfich

Función: Especifica un controlador opcional de dispositivo que hay que cargar durante la configuración del sistema.

Notas: Los parámetros del comando definen el fichero en el que está el controlador de dispositivo especificado.

Algunos de los controladores de dispositivo son:

— CON (teclado y monitor).

— COM1 (AUX), COM2, COM3, COM4 (puertos adaptadores para la comunicación asíncrona).

— LPT1 (PRN), LPT2, LPT3 (impresoras paralelas).

— Unidades de disco flexible («diskettes») y de disco rígido («disco duro»).

— CLOCK$ (reloj de tiempo real).

Estos controladores se cargarán automáticamente durante la configuración del sistema; por tanto, no es necesario incluir ningún comando DEVICE en el fichero CONFIG.SYS con estos dispositivos estándar.

FILES =

Formato: FILES = nn

Función: Determina el número máximo de ficheros que pueden abrirse simultáneamente durante el funcionamiento del sistema.

Notas: El parámetro nn puede tomar cualquier valor de 1 a 99. El valor por defecto es 8.

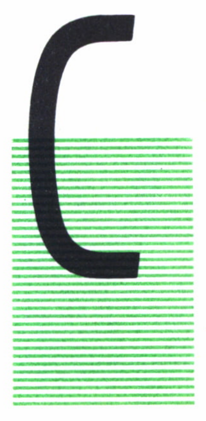
SHELL =

Formato: SHELL = [u:] [ruta] nombfich

Función: Especifica el intérprete alternativo de comandos que hay que cargar en lugar del intérprete estándar de comandos COMMAND.CON.

Notas: Los parámetros del comando definen el fichero que contiene el intérprete de comandos deseado.

INTRODUCCION AL CP/M 



P/M es un sistema operativo en «diskette» para microordenadores. Hay disponibles versiones de CP/M para una amplia variedad de microordenadores. CP/M-80 puede usarse en casi todos los microordenadores que usan unidad central de tratamiento (microprocesador) 8080 y tienen unidades de disco flexible («diskette» de 8 ó 5 1/4 ". CP/M-86 puede usarse en casi todos los microordenadores que utilizan unidad central de tramiento 8086 u 8088 y tienen unidades de disco flexible.



TIPOS DE CP/M

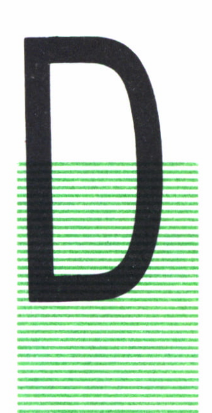
Puede considerarse que CP/M es un sistema operativo casi estándar en el campo de los microordenadores, pero no todos los CP/M son iguales. CP/M-80 y CP/M-86 varían según las instrucciones de E/S (entrada/salida) de cada máquina y también pueden variar por otros motivos.

Puesto que CP/M fue diseñado originalmente para sistemas de "diskettes", requiere cambios para usar unidades de disco duro. Esta es la diferencia primordial entre las versiones 1.4 y 2.2 de CP/M-80.

Los diferentes tipos de CP/M-80 y CP/M-86 reflejan también el gran número de firmas de ordenadores que usan CP/M. Cada una puede añadir programas de utilidad o refinamientos a CP/M para aumentar la eficiencia de una máquina particular.

COMANDOS DEL CP/M 13

COMANDOS PARA EL MANTENIMIENTO DE DISCOS



IR

Función: Muestra en la pantalla los nombres y otra información relativa a los ficheros accesibles al usuario en la unidad implícita o en la especificada, a excepción de los ficheros del sistema.

Formatos:

— DIR

Muestra los nombres y distintivos de tipo de todos los ficheros que hay en el «diskette» de la unidad implícita.

— DIR unidad

Muestra los nombre y distintivos de tipo de todos los ficheros que hay en el «diskette» de la unidad que se especifique.

— DIR fichero

Indica si el fichero especificado está en el «diskette».

Ejemplo:

A > DIR B :

Muestra los nombres y distintivos de todos los ficheros que hay en el «diskette» de la unidad B.

DIRSYS

Función: Muestra en la pantalla los nombres y otra información relativa a los ficheros de sistema accesibles al usuario.

Formato: Análogos a los del comando DIR.

Ejemplo:

A > DIRSYS B:

— Muestra los nombres y distintivos de tipo de todos los ficheros de sistema que hay en el «diskette» de la unidad B.

ERASE

Función: Borra del directorio la reseña del fichero (o ficheros) especificado/s.

Formato:

ERASE unidad : fichero

— Borra el fichero especificado del «diskette» de la unidad especificada.

ERASE fichero

— Borra el fichero especificado del «diskette» de la unidad implícita.

Ejemplo:

A > ERASE PROG1.ASM

— Borra el fichero PROG1.ASM del «diskette» de la unidad A (implícita).

A > ERASE B : PRO.PAS

— Borra el fichero PRO.PAS del «diskette» de la unidad B.

GET

Función: Ordena al sistema que en el futuro lea en el fichero especificado todas las entradas que normalmente captaría por consola. Tales entradas pueden ser órdenes de CP/M, datos requeridos por los programas o ambas cosas.

Formato:

GET CONSOLE INPUT FROM FILE fichero

— Ordena al sistema que lea en el fichero especificado las entradas al programa que se va a ejecutar a continuación. La orden que pone en marcha el programa tiene que ser la siguiente que se dé por el teclado.

GET CONSOLE INPUT FROM CONSOLE

— Ordena al sistema que vuelva a captar las entradas por el teclado.

Ejemplo:

A > GET CONSOLE INPUT FROM FILE ARCHI1.DAT

A > MIPROG

— Ordena al sistema que lea en el fichero ARCHI1.DAT de la unidad implícita todas las entradas al programa MIPROG, que de otra forma captaría por el teclado.

INITDIR

Función: Este comando se utiliza en combinación con SET para habilitar la estampación de fecha y hora de los ficheros grabados en el «diskette» de la unidad especificada. INITDIR reinicializa el directorio

Formato:

INITDIR unidad:
— Reinicializa el directorio del «diskette» que está en la unidad especificada para habilitar la estampación de fecha y hora o bien para suprimir esta información si ya está habilitada.

Ejemplo:

A > INITDIR B:
— Reinicializa el directorio del «diskette» que está en la unidad B.

PIP

Función: Transfiere datos del dispositivo lógico de entrada (fuente) al dispositivo lógico de salida (destino). Por ejemplo, puede servir para copiar ficheros de un «diskette» a otro, para listar por la impresora el contenido de un fichero de texto, etc.

Además, puede combinar las entradas tomadas de varias fuentes para producir una sola salida.

Formatos:

PIP fichero destino = fichero - fuente
— Copia el fichero fuente y guarda la copia en el fichero destino.

PIP fichero destino = unidad :
— Copia el fichero que hay en el «diskette» especificado y cuyos nombre y distintivo de tipo coinciden con los especificados en el fichero destino guardando la copia en el «diskette» destino, con el mismo nombre y el mismo distintivo de tipo.

PIP unidad : = fichero fuente
— Copia el fichero fuente especificado y guarda la copia en el «diskette» destino con el mismo nombre y el mismo distintivo de tipo.

PIP fichero destino = fuente-1, fuente-2 [,fuente-n]
— Combina los ficheros fuente en el orden especificado y guarda el resultado en el fichero especificado.

Todas estas variantes de PIP pueden ser ampliadas mediante las opciones siguientes:

OPCIONES DE PIP

Opción	Resultado
C	Pide confirmación antes de realizar cada operación de copia.
E	Los datos transferidos son reproducidos en la pantalla.
F	Suprime todos los códigos de avance de hoja que haya en el fichero fuente.
G*n*	Identifica el fichero como perteneciente al usuario *n*.
L	Convierte todas las letras mayúsculas en minúsculas.
N	Añade números de línea al fichero destino.
O	Hace que se transfiera el fichero entero, evitando así que los caracteres no imprimibles sean interpretados por PIP como marcadores de fin de fichero.
P*n*	Establece en *n* el número de líneas por página en el fichero destino.
Q*cadena*'control-Z'	El proceso de copia concluye en cuanto se termina de copiar la *cadena* especificada.
R	Permite buscar en el directorio de ficheros de sistema, cosa que PIP no hace normalmente.
S*cadena*'control-Z'	El proceso de copia empieza por el principio de la *cadena* especificada.
U	Convierte todas las letras minúsculas en mayúsculas.
V	Compara el fichero destino con el fichero origen para comprobar que la copia es correcta.
W	Permite que al grabar la copia es escriba sobre un fichero de «sólo lectura» que pueda haber con la misma especificación que el fichero destino.
Z	Pone a cero todos los bits de paridad de los datos del fichero destino.

Ejemplos:

A > PIP CAP1.TXT = SEC1.TXT,SEC2.TXT,SEC3.TXT

— Combina el contenido de los ficheros SEC1.TXT, SEC2.TXT, SEC3.TXT y guarda el resultado en un fichero al que da el nombre de CAP1.TXT . Todos los fichero están en la unidad implícita (unidad A).

A > PIP LST : = CON : [U N P40]

— Lista por la impresora todo lo que se escribe en el teclado; cambia de página cada 40 líneas; convierte las minúsculas en mayúsculas; numera las líneas.

PUT

Función: Ordena al sistema que en el futuro escriba en el fichero especificado todas las salidas que normalmente enviaría a la consola (pantalla) o a la impresora, hasta que se le ordene otra cosa o hasta que finalice el programa que se va a ejecutar a continuación.

Formato:

PUT CONSOLE OUTPUT TO FILE fichero
— Ordena al sistema que dirija al fichero especificado la salida por la consola. La salida será reproducida en pantalla.

PUT PRINTER OUTPUT FILE fichero
— Ordena al sistema que dirija al fichero especificado la salida por la impresora. La salida no será reproducida en la impresora.

PUT CONSOLE OUTPUT TO CONSOLE
— Ordena al sistema que envíe solamente a la consola la salida que normalmente debe dirigir a ella.

PUT PRINTER OUTPUT TO PRINTER
— Ordena al sistema que envíe solamente a la impresora la salida que normalmente debe dirigir a ella.

Ejemplo:

A > PUT CONSOLE OUTPUT TO FILE ARCHI1.TXT
— Ordena al sistema que escriba en el fichero ARCHI1.TXT de la unidad implícita todas las salidas que de otra forma enviaría a la pantalla.

RENAME

Función: Cambia el nombre de uno o varios ficheros en sus correspondientes reseñas en el directorio de un «diskette».

Formato:

RENAME fichero - nuevo = fichero - antiguo
— Cambia el nombre y distintivo de tipo del fichero antiguo por los especificados para el fichero nuevo.

RENAME
— Realiza el cambio de nombre preguntando al usuario los nombres y distintivos de tipo de los ficheros nuevo y antiguo.

Ejemplo:

A > RENAME PROG1.TXT = PROG2.TXT
— Da el nuevo nombre PROG1.TXT al fichero PROG2.TXT .

SET

Función: Asigna atributos a los ficheros y a las unidades y permite etiquetar los «diskettes» para facilitar su catalogación.

Formatos:

SET fichero [opción [,opción]]
— Da al fichero mencionado los atributos especificados por las opciones. Las opciones pueden ser:

— DIR convierte un fichero de sistema en un fichero ordinario.
— SYS convierte un fichero ordinario en un fichero de sistema.
— RO hace que el fichero sólo sea accesible para su lectura.
— RW hace que el fichero sólo sea accesible tanto para lectura como para escritura.

SET unidad : [opción]
— Da a la unidad especificada el atributo especificado por la opción. Las opciones pueden ser:

— RO permite el acceso a los ficheros de la unidad solamente para lectura.
— RW permite el acceso a los ficheros de la unidad tanto para lectura como para escritura.

SET unidad : [NAME = nombre - etiqueta]
— Asigna la etiqueta especificada al «diskette» que está en la unidad especificada.

Ejemplos:

A > SET MIFICH.TXT [RO,SYS]
— Convierte el fichero MIFICH.TXT en un fichero de sistema al cual sólo se puede acceder para leerlo.

A > SET B : [RO]
— Limita el acceso a la unidad B, permitiendo solamente la lectura en ella.

SHOW

Función: Muestra la siguiente información sobre los discos que han sido reconocidos por el sistema: modo de acceso, espacio libre en el disco, etiqueta del disco, número de fichero que hay en cada zona de usuario y número de reseñas que quedan libres en el directorio.

Formatos:

SHOW
— Muestra el modo de acceso y el espacio libre para cada unidad reconocida por el sistema.

SHOW unidad:
— Muestra el modo de acceso y el espacio libre para el «diskette» que está en la unidad especificada.

SHOW unidad: [USERS]
— Detalla todos los números de usuario que se han utilizado en el «diskette», el número de ficheros que hay en cada zona de usuario y el número de reseñas que quedan libres en el directorio del «diskette».

SHOW unidad: [DIR]
— Muestra el número de reseñas que quedan libres en el directorio del «diskette»

SHOW unidad: [DRIVE]
— Muestra las características de la unidad especificada.
Ejemplo:

A > SHOW [USERS]
— Este comando dará como respuesta:

Active User:	1
Active Files:	0 2 3 4
A : ı Of Files:	10 2 11 1
A : Number Of Free Directory Entries: 28	

SUBMIT

Función: Hace que se ejecute la serie de órdenes que está grabada en un fichero de tipo SUB, igual que si las escribiese una a una en el teclado.
El fichero de tipo SUB puede contener órdenes de CP/M, órdenes SUBMIT anidadas y datos de entrada para órdenes de CP/M o para programas.
Formatos:

SUBMIT fichero
— Hace que se ejecute una a una las órdenes grabadas en el fichero especificado, que ha de ser de tipo SUB.

SUBMIT
— Espera que se introduzca por el teclado el nombre del fichero cuyas líneas van a ser ejecutadas, así como otras posibles entradas.
Ejemplo:

A > SUBMIT FICHSUB.SUB
— Hace que se ejecute la serie de órdenes grabadas en el fichero FICHSUB.SUB.

TYPE

Función: Muestra el contenido de un fichero ASCII, bien pantalla a pantalla, bien de forma continuada.
Formatos:

TYPE fichero
— Muestra el contenido del fichero especificado, pantalla a pantalla

TYPE fichero [NOPAGE]
— Muestra el contenido del fichero especificado, de forma continua.

USER

Función: Establece el número de usuarios; en lo sucesivo, a todos los ficheros creados se les asigna ese número y sólo se puede acceder a los ficheros de esa misma área de usuario y a los del área 0.

El número de usuario puede tener cualquier valor entre 0 y 15.
Formato:

USER número
— Cambia el número actual de usuario al número especificado. Si no se especifica ninguno, aparece un mensaje que lo solicita.
Ejemplos:

A > USER 3
— Cambia el número de usuario actual (que es el implícito, 0) a 3.



COMANDOS PARA LA PREPARACION DEL HARDWARE

DATE

Función: Establece y muestra la fecha y la hora.
Formatos:

DATE
— Muestra la fecha y hora actuales.

DATE MM/DD/AA hh:mm:ss
— Pone en hora el reloj.

LANGUAGE

Función: Selecciona uno de los ocho juegos de caracteres internacionales.
Formato:

LANGUAGE n
— Selecciona el juego de caracteres número n.
Identificadores del juego de caracteres:

— 0 EEUU
— 1 FRANCIA
— 2 ALEMANIA
— 3 U.K.

— 4 DINAMARCA
— 5 SUECIA
— 6 ITALIA
— 7 ESPAÑA

PALETTE

Función: Cambia el modo de visualización en pantalla, de caracteres claros sobre fondo oscuro a la situación inversa, y viceversa.
Formatos:

PALETTE 1 0
— Establece vídeo inverso (caracteres oscuros sobre fondo claro).

PALETTE 0 1
— Establece vídeo normal.

SET24X80

Función: Selecciona la pantalla de 24 líneas por 80 columnas.
Formatos:

SET24X80 ON
— Selecciona el modo 24 x 80.

SET24X80 OFF
— Restaura las características normales de la pantalla.



COMANDOS PARA LA PROGRAMACION AVANZADA

DUMP

Función: Visualiza en la pantalla el contenido de un fichero, en forma hexadecimal y ASCII.
Formato:

DUMP fichero
— Visualiza el contenido del fichero especificado en la forma indicada.

ED

Función: Inicia la edición del fichero especificado. También puede servir para crear un fichero nuevo.
Formato:

ED fichero
— Abre el fichero especificado. Si no encuentra el fichero especificado, creará un nuevo fichero con dicho nombre.

ORDENES DE ED

Orden	Acción
*n*A	Añade *n* líneas al espacio de trabajo de ED tomándolas del fichero fuente.
0A	Añade líneas, tomadas del fichero fuente, al espacio de trabajo hasta que se llena la mitad de éste.
#A	Añade líneas, tomadas del fichero fuente, al espacio de trabajo hasta que éste se llena o se llega al final del fichero fuente.
B, -B	Lleva el puntero de texto al principio (B) o al final (-B) del espacio de trabajo.
*n*C	Hace avanzar el puntero *n* posiciones (o retroceder, si *n* es negativo).
*n*D	Borra los *n* caracteres siguientes al puntero (o los anteriores, si *n* es negativo).
E	Graba el fichero nuevo y retorna a CP/M.
F*cadena*'control-Z'	Busca la cadena de caracteres especificada en el espacio de trabajo, a partir de la posición señalada por el puntero.
H	Grava la versión actual del fichero y reinicia la edición, tomando este fichero como fuente.
i	Activa el modo de inserción. Para desactivarlo se pulsa SAL.
i*cadena*'control-Z'	Inserta la cadena de caracteres en la posición del puntero.
*n*K	Borra *n* líneas a partir de la posición del puntero hacia delante (o hacia atrás, si *n* es negativo).
*n*L	Hace avanzar el puntero *n* líneas por el espacio de trabajo (o retroceder, si *n* es negativo).
OL	Lleva el puntero al principio de la línea actual.
*n*M*órdenes*	Hace que ED ejecute *n* veces las órdenes dadas.
n	Hace avanzar (o retroceder) el puntero *n* líneas y muestra en la pantalla la línea a la que llega.
n:	Lleva el puntero a la línea número *n* y la muestra en la pantalla.
O	Ignora los cambios introducidos en la edición y vuelve a la versión original para reeditarla.
*n*P	Hace avanzar (o retroceder) el puntero por el espacio de trabajo *n* líneas y muestra en la pantalla todas las líneas recorridas.
Q	Abandona la edición y retorna a CP/M.
R*fichero*'control-Z'	Carga el fichero en el espacio de trabajo, a continuación de lo que ya haya en éste.
*n*Scadena-antigua 'control-Z' cadena-nueva	realiza *n* veces la operación de buscar la cadena antigua y sustituirla por la nueva, partiendo de la posición actual del puntero.
*n*T	Muestra en la pantalla las *n* líneas siguientes (o anteriores, si *n* es negativo) al puntero, dejando éste en la misma posición.
*n*W	Escribe las *n* primeras líneas del espacio de trabajo en el fichero destino.
*n*X*fichero*	Escribe *n* líneas del espacio de trabajo en el fichero especificado, poniéndolas a continuación de las que ya se hayan escrito en él mediante una orden *n*X*fichero* anterior.

LINK

Función: Combina módulos objeto reubicables para formar un fichero de orden.

SAVE

Función: Intercepta el retorno al sistema al terminar la ejecución de un programa y luego copia el contenido de la memoria, entre dos direcciones hexadecimales especificadas, en un fichero.

La orden se pone en marcha escribiendo SAVE.

HELP

Función: Da información técnica en forma abreviada sobre las órdenes de CP/M.

Al ejecutar este comando pide el tema sobre el que se desea recibir la información.

Cuando usted compra un ordenador, casi con toda seguridad que lo primero que deseará será verlo en funcionamiento. Pero no basta con accionar al interruptor de alimentación para que el ordenador comience a trabajar. Veamos por qué. Cuando usted, por ejemplo desea ejecutar un programa, éste debe estar cargado en la memoria del ordenador. Es evidente que dentro del ordenador debe existir «algo» que le cargue su programa. Este «algo» no es ni más ni menos que un gran programa (¿pensaba usted que su máquina tenía "duendes"?) llamado **SISTEMA OPERATIVO.**

Como este gran programa superaría, posiblemente, la capacidad de la memoria de su ordenador, el Sistema Operativo se compone de un conjunto de programas que le ayudan a gestionar los recursos de su ordenador (memoria, pantallas, discos, etc.).

Este libro pretende introducir al lector en el apasionante y complejo mundo de los Sistemas Operativos, permitiéndole gestionar más eficazmente su ordenador, ahorrándole esfuerzo en el manejo de su máquina.